Erkki Mäkinen (toim.)

Tietojenkäsittelytieteellisiä tutkielmia
Syksy 2015

INFORMAATIOTIETEIDEN YKSIKKÖ
TAMPEREEN YLIOPISTO

INFORMAATIOTIETEIDEN YKSIKÖN RAPORTTEJA 42/2015

TAMPERE 2015

TAMPEREEN YLIOPISTO
INFORMAATIOTIETEIDEN YKSIKKÖ
INFORMAATIOTIETEIDEN YKSIKÖN RAPORTTEJA 42/2015
JOULUKUU 2015

Erkki Mäkinen (toim.)

Tietojenkäsittelytieteellisiä tutkielmia
Syksy 2015

INFORMAATIOTIETEIDEN YKSIKKÖ
33014 TAMPEREEN YLIOPISTO

ISBN 978-952-03-0032-6 (pdf)

ISSN-L 1799-8158
ISSN 1799-8158

Sisällysluettelo

Neuroverkot kuvien luokittelijoina……..…….……..1
Lauri Hursti
Julkisiin tiloihin sijoitettujen suurikokoisten, vuorovaikutteisten näyttöjen tarkastelua käytettävyyden näkökulmasta……..……..……...…….21
Henri Juvonen
Yksinkertaiset hierarkkiset klusterointimenetelmät……..…….……39
Valtteri Kostiainen
Pervasiivisten pelien tarjoamat mahdollisuudet opetuskäytössä…….……58
Ville Lahtinen
Käyttäjähyväksynnän malleista.…….…….…….……76
Merita Lemmetty
Software product line engineering -paradigman testauksen ongelmat….……..….93
Jarmo Multanen
K-means -klusterointialgoritmi ja sen variaatiot ……..…108
Miikka Mäkipörhölä
Correct-by-construction-menetelmät ohjelmistotuotannossa..……...……128
Mikko Paukkonen
Junan matkustajainformaatiojärjestelmän järjestelmätestaus..……...……144
Konsta Peltola
Antipatternit ja antipatternien havainnointi..…….……..…163
Atte Pietikäinen
Terveyssovellukset ja niiden käyttäjäkokemus……. ……...……176
Emma Salminen
Katsaus valvontakamerajärjestelmien yksityisyyden suojausmenetelmiin ……..…196
Anniina Seppä
Harmoninen haku metaheuristisena algoritmina…….……...……..……217
Andreas Valjakka
Pariohjelmoinnin käyttö opetuksessa ……...……...237
Marjut Vornanen

Neuroverkot kuvien luokittelijoina

Lauri Hursti

Tiivistelmä

Neuroverkoilla voidaan algoritmisesti luokitella mitä tahansa numeerisia syötteitä. Menetelmän vahvuus on kyky oppia luokittelun olennaiset piirteet itse, mikä tekee siitä toimivan monille sovellusalueille. Konenäössä neuroverkkoja ollaan käytetty vaihtelevalla menestyksellä. Vaikeimmissa tehtävissä kehitys on vielä kesken, mutta viime vuosina saavutettu parannus luokittelutarkkuudessa on lupaavaa. Esittelen tässä tutkielmassa neuroverkkojen käyttöä ja tutkimusta kuvantunnistuksen kannalta.

Avainsanat: *neuroverkot, koneoppiminen, konenäkö, kuvien luokittelu*

1. Johdanto

> *"Pulut eivät vaikuta älyn jättiläisiltä, mutta ne osaavat erottaa syövän koepalasta otetusta kuvasta paremmin kuin tietokoneohjelmat."*
>
> -Helsingin Sanomat, 20.11.2015

Ihmisinä näemme ja analysoimme näkemäämme nopeasti ja tarkasti. Pystymme muutamassa sadassa millisekunnissa rakentamaan kattavan kuvan näkemästämme, ja tämä onnistuu ilman suurempia tietoisia ponnisteluita [Behnke 2003]. Neuroverkko on aivotutkimuksesta innoituksensa saanut koneoppimisen menetelmä tehdä vastaavaa syötteiden luokittelua algoritmisesti.

Jotta voisimme tunnistaa ja erotella syötteitä, pitää syötteen luokittelun kannalta oleelliset piirteet olla jollain tasolla tiedossa. Erotamme vaivattomasti esimerkiksi numerot kolme ja viisi. Piirre-eroja voisi yrittää formalisoida: numero kolme on kupera oikealle, numeron viisi vasemmassa ylänurkassa on terävä kulma, jne. Tässä luokittelutehtävässä oleellista olisi ainakin etsiä kulmia ja kaaria eri puolilta kuvaa. Paikannettujen kaarteiden ja kulmien avulla voisimme tehdä päätelmiä syötteen luokasta.

Esineiden ja olioiden kohdalla oleellisten piirteiden määrä kasvaa suuremmaksi, ja erot luokkien välillä ovat vaikeampia formalisoida. Samaan luokkaan kuuluvien syötteiden välillä on enemmän vaihtelua, koska osa piirteistä on piilossa ja tai niiden mitattavat ominaisuudet vaihtelevat luonnollisissa tilanteissa. Ihmisnäköä tämä variaatio ei haittaa: tunnistamme sivulta näkyvän punaisen auton hämärässä samaksi autoksi, jonka näimme päivänvalossa takaa. Näkyvissä olevat muodot ja valon aallonpituus vaihtelevat, mutta tunnistaminen onnistuu tästä huolimatta [Behnke 2003].

Piirteiden keräämiseksi tarkoitetun ohjelman kirjoittaminen käsin on erittäin vaikeaa ja pitäisi tehdä jokaisen ongelman kanssa erikseen [LeCun *et al.* 1998a]. Neuroverkot ja muut koneoppimisen menetelmät yrittävät kiertää käsinkirjoitettujen piirteiden ongelman. Laajan harjoitussyötekokoelman avulla neuroverkko *oppii*, mitkä piirteet ovat syötteen analysoinnissa olennaisia ja mitkä piirre-erot yksilöivät tunnistettavia luokkia.

Luvussa 2 esittelen lyhyesti neuroverkkojen peruskäsitteet. Kuvaan, mitä ovat verkon perusyksiköt neuronit ja millaisista osista verkko yleensä muodostuu. Luvussa 3 tarkastelen erityisesti kuvantunnistuksessa käytettyjä verkkoja. Luvussa esitellään kuvantunnistukselle tyypillinen suodattava neuroverkko ja katsotaan, millaisia käytännön sovelluksia neuroverkoilla on toteutettu konenäössä. Lopuksi, luvussa 4, käsittelen neuroverkkojen oppimista kuvantunnistuksen kannalta.

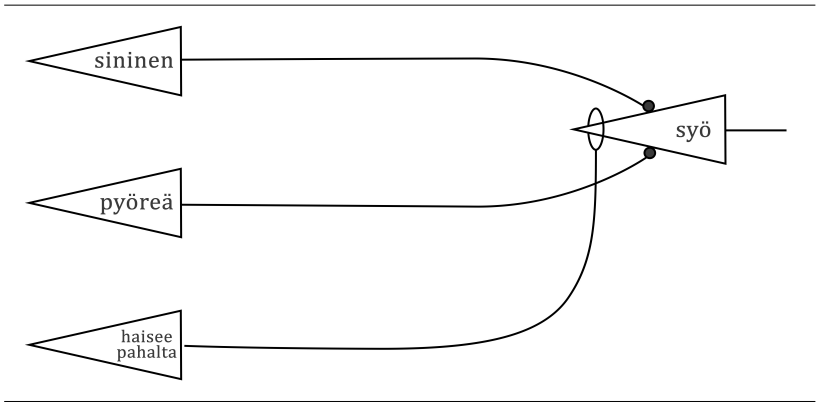
Neuroverkoilla tällä hetkellä suoritettavat käytännön tehtävät vaihtelevat käsinkirjoitettujen numeroiden lukemisesta mikroskooppisten kuvien analysointiin ja liikennemerkkien tunnistamiseen. Esineiden ja olioiden tunnistamisessa jäädään vielä kauas ihmisen tai pulun tarkkuudesta. Vaikeissa luokittelutehtävissä toimivat neuroverkot vaativat siis edelleen kehittämistä. Tautien diagnosointi ja avustettu näkö liikenteessä ovat esimerkkejä kriittisistä sovellusalueista, joille paremmista neuroverkoista voisi olla hyötyä.

2. Neuronien muodostamat verkot - peruskäsitteet

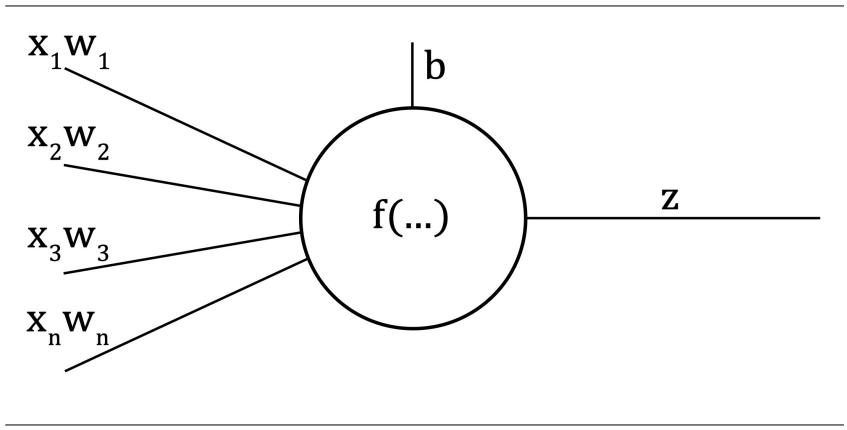
2.1 Neuroni

Neuraalilaskennan keksijät McCulloch ja Pitts [1943] kuvaavat, miten aistiärsykkeet verkottuneiden loogisten porttien kautta muuttuvat havainnoiksi ja toiminnan valinnoiksi. Esimerkissä ärsykkeet ”pyöreä” ja ”tummansininen” aiheuttavat yhdessä reaktion ”syö” (kuva 1). Jos vain toinen ärsykkeistä toteutuu, ei syömistä tapahdu. Ehkäpä tarkkailun alaisena oleva esine ei olekaan mustikka. Kiihottavien ärsykkeiden lisäksi neuroni voisi saada estäviä ärsykkeitä, jollainen on esimerkin ”haisee pahalta”. Esimerkin neuroni laukeaisi vain, jos se saa kaksi kiihottavaa ärsykettä eikä yhtään estävää. McCulloch ja Pitts olivat ensimmäiset, jotka sovelsivat logiikan ja matemaattisen laskennan menetelmiä käytöksen selittämiseen [Piccinini 2004].

Nykyaikaiset verkot operoivat reaaliluvuilla totuusarvojen sijaan. Verkon perusyksikkö on neuroni (kuva 2). Neuronin abstrakti tehtävä on tutkia sitä edeltävässä kerroksessa tapahtuvaa aktiivisuutta ja tehdä edeltävän tason toiminnan perusteella päätös itsensä aktivoinnista. Kuvassa neuronia edeltävän tason aktiivisuutta kuvaa arvot x_0-x_n. Edellisen kerroksen yksiköt ovat yhteydessä neuroniin omalla yhteydellään, joilla kullakin on tietty paino w_o-w_n. Aktiviteetit ovat useimmiten positiivisia ja painot sekä positiivisia että negatiivisia reaalilukuja. McCullochin ja Pittsin tapaan yhteys edelliseen kerroksen yksikköön voi siis olla neuronin aktiivisuutta joko kiihottava tai rajoittava. Negatiivinen paino vastaa rajoittavaa ja positiivinen kiihottavaa vaikutusta.



Kuva 1. McCulloch-Pitts -verkko, jossa mustaan ympyrään päättyvät yhteydet ovat kiihottavia ja lassoon päättyvät rajoittavia.



Kuva 2. Neuroni.

Neuroni laskee edellisestä kerroksesta tulevien aktiviteettien ja painojen tulojen summan ja lisää summaan kallistuman (*bias*) *b.* Tämän jälkeen neuroni päättää jonkin aktivointifunktion *f* avulla, mikä on eteenpäin lähetettävä aktiviteetti *z*. Kaavana ilmaistuna neuronin aktiviteetti olisi

$$z = f\left(b + \sum_{i=0}^{n} w_i x_i\right). \quad (1)$$

Aktivointifunktio *f* on vaihdellut neuroverkkojen kehityksen myötä. Totuusarvoja kuvaava binäärinen askelfunktio, jossa *z* saa vain arvoja 0 tai 1 ei sovellu nykyisiin neuroverkkoihin oppimismenetelmien vuoksi: funktion pitää olla derivoitavissa kaikissa kohdissa [Rumelhart *et al.* 1986]. Tarkastelen aktivointifunktion merkitystä neuroverkolle tarkemmin kohdassa 4.4.

2.2 Neuroverkko

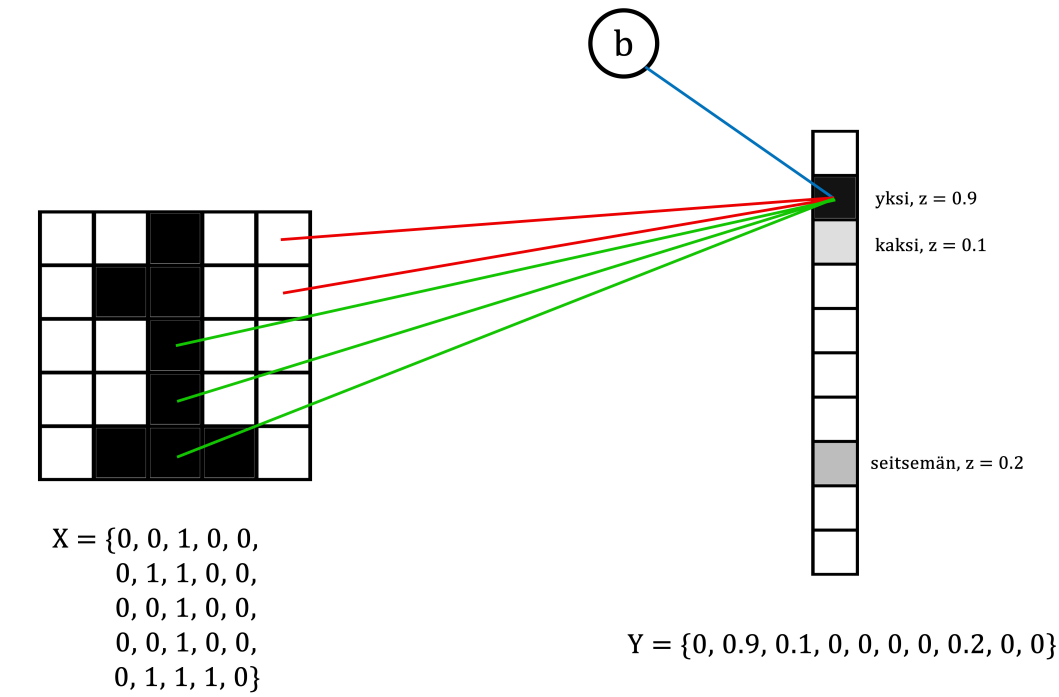
Yksittäinen neuroni ei pysty kovinkaan monimutkaisiin tehtäviin. Se voi toimia tietynlaisen piirteen esiintymisen tarkkailijana, mutta käytännöllisempiä sovelluksia saadaan, kun monta erilaista neuronia yhdistetään verkoksi. Verkko vaatii vähintään jonkinlaiset syöte- ja tuloskerrokset, jotka toimivat verkon näkyvänä käyttöliittymänä. Yksinkertai-

seen verkkoon riittäisi pelkkä syöte, tulos ja niiden väliset yhteydet, mutta väliin voidaan sijoittaa vaihteleva määrä piilotettuja kerroksia.

Syötekerros on jonkinlainen vektori, joka kuvaa syötteen numeerisesti. Mustavalkoista kuvaa vastaavassa vektorissa kukin vektorin komponentti olisi yhden pikselin tummuus. Kuvan syötevektori voidaan ajatella kaksiulotteisena taulukkona [LeCun *et al.* 1989]. Värikuvan RGB-arvot laajentavat syötesäiliön kolmiulotteiseksi taulukoksi [Krizhevsky *et al.* 2012]. Mikä tahansa kuvasyöte voitaisiin järjestää yksiulotteisesti, mutta syötteen komponenttien järjestäminen moniulotteiseen taulukkoon on käsitteellisesti intuitiivisempaa ja helpottaa työtä suodattavien verkkojen parissa. Moniulotteinen syötetaso ottaa huomioon kuvan kaksiulotteisen luonteen ja paikalliset yhteydet [LeCun *et al.* 1998a].

Tuloskerros puolestaan on useimmiten kokoelma neuroneita, jossa kunkin neuronin aktiivisuus kertoo syötteen kuuluvan tiettyyn luokkaan. Väärien luokkien saaman kannatuksen avulla voidaan arvioida luokittelun luotettavuutta.

Yksinkertaisella kaksikerroksisella verkolla saadaan yhdistettyä toisiaan muistuttavat syötteet samoihin luokkiin [Rumelhart *et al.* 1986]. Otetaan esimerkiksi kaksikerroksinen neuroverkko, joka tunnistaa pikseleistä muodostuvia numeroita (kuva 3)**.** Syöte X olisi tyypillisesti kuvan pikselien intensiteetit esitettynä vektorina, ja Y vektori, joka kuvaa neuroverkon arvausta syötteen luokasta. Tulosvektori koostetaan kaikista tulostason neuroneiden aktiviteeteistä. Esimerkissä kallistuma *b* on lisätty jokaiseen tulosneuroniin omana aktiviteettinään, joka saa arvon yksi. Kallistuman ja neuronin välinen paino hallitsee kallistuman suuruutta. Kun kallistumaa säätelee käytännössä yksi paino, tulee oppimisesta yksinkertaisempaa.

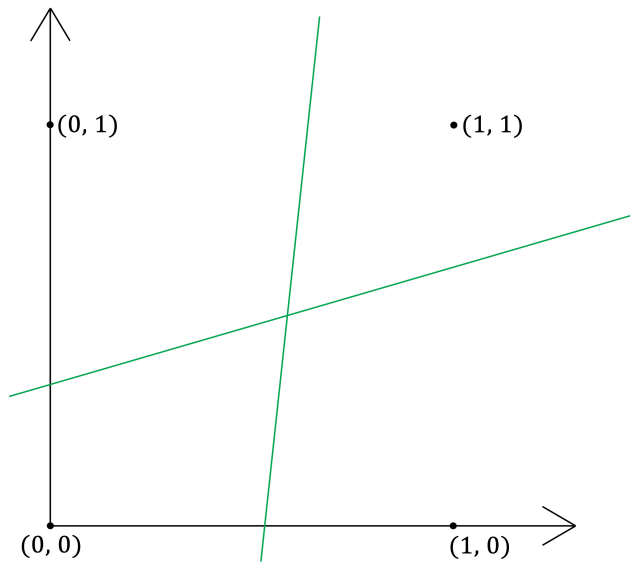


Kuva 3. Yksinkertainen neuroverkko numeroiden tunnistamiseen.

Jokainen syötevektorin komponentti on yhteydessä kaikkiin tuloskerroksen neuroneihin. Selkeyden vuoksi vain muutama yhteys neuroniin ”yksi” on piirretty näkyviin. Hyvin luokittelevalla verkolla vihreällä merkityt painot syötevektorista olisivat todennäköisesti positiivisia. Ne vahvistaisivat neuronin ”yksi” aktiivisuutta. Vastaavasti punaiset painot olisivat todennäköisesti negatiivisia tai lähellä nollaa. Tuloskerroksesta huomataan, että neuroverkko ei ole puhtaasti luokan ”yksi” kannalla. Myös luokat ”kaksi” ja ”seitsemän” ovat saaneet kannatusta. Neuroverkko on aina enemmänkin arvio kuin tarkka esitys syötteitä täydellisesti luokittelevasta funktiosta.

Tällainen yksinkertainen verkko onnistuu luokittelemaan samaan luokkaan syötteitä, jotka muistuttavat toisiaan. Piilotettujen kerrosten avulla voidaan teoriassa luokitella mitkä tahansa syötteet mielivaltaisiin luokkiin riippumatta samaan luokkaan kuuluvien syötteiden samanlaisuudesta [Rumelhart *et al.* 1986]. Voidaan myös puhua lineaarisesti riippumattomista joukoista. Kuvitellaan joukko syötteitä, joita kuvaa n-ulotteiset vektorit n-ulotteisessa avaruudessa. Jos joukot ovat lineaarisesti riippumattomia, tähän avaruuteen ei voida sijoittaa hypertasoa, joka jakaisi avaruuden syötepisteet kahteen eri luokkaan. Klassinen esimerkki tällaisesta tilanteesta on loogisen XOR-operaattorin toiminta (kuva 4), johon Rumelhart ja muut tarjoavat myös yhteen piilotettuun kerrokseen perustuvan ratkaisun. Kuten kuvasta nähdään, tuloksen 1 tuottavat syötteet 0,1 ja 1,0 eivät ole yhdellä suoralla erotettavissa omaksi luokakseen syötteistä 0,0 ja 1,1.

syöte1	syöte2	tulos
0	0	0
1	0	1
0	1	1
1	1	0



Kuva 4. XOR-operaation totuustaulu ja syötepisteet koordinaatistossa. Vihreät suorat eivät saa tuloksen perusteella jaettua syötteitä kahteen luokkaan.

XOR-ongelmassa mahdollisten syötevektoreiden joukko on tunnettu. Neljällä toistuvalla harjoitusesimerkillä neuroverkko saadaan oppimaan sopivat painot oikeaa luokittelua varten. XOR-verkon ei tarvitse yleistyä luokittelemaan harjoitushetkellä tuntemattomia syötevektoreita. Tilanne on toinen useimmissa käytännön tehtävissä. Luon-

nollisten valokuvien ja käsinkirjoitettujen merkkien tapauksessa syötevektorien joukko ei ole ennalta tunnettu, mikä vaikeuttaa luokittelutehtävää. Saatavilla olevien esimerkkien avulla neuroverkon pitää oppia reagoimaan oikein myös syötevektoreihin, joita ei vielä tunneta. Tähän mennessä parhaiten toimiva ratkaisu on suodattavat neuroverkot, joita käsittelen seuraavaksi luvussa 3.

3. Kuvia suodattavat verkot

3.1 Paikallisen riippuvuuden ongelma – suodattavat verkot ratkaisuna

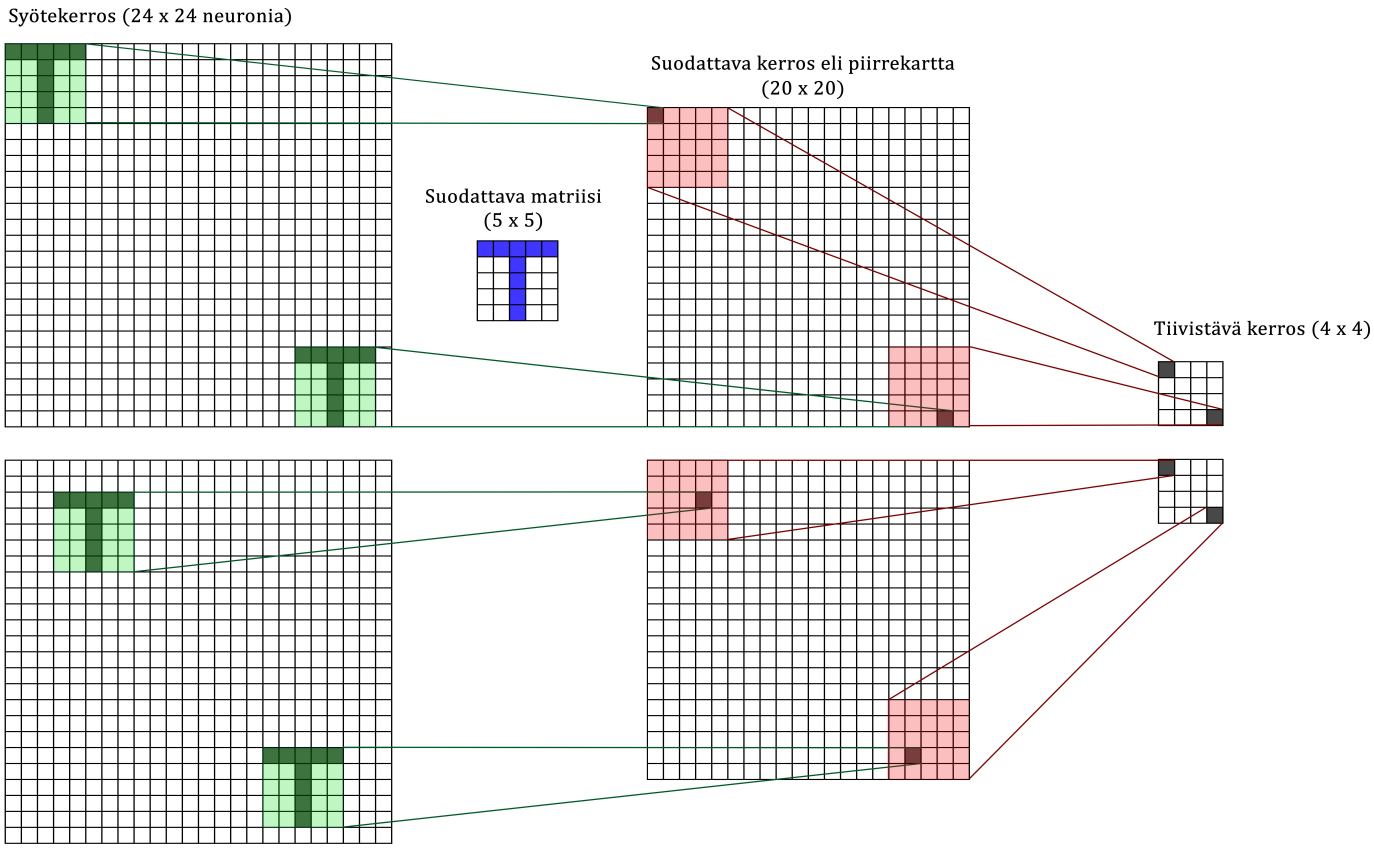
Kuvassa 3 esitetty neuroverkko toimii täydellisesti, jos samaan luokkaan kuuluvat syötevektorit muistuttavat toisiaan. Se voitaisiin opettaa lukemaan tarkasti esimerkiksi tietyllä fontilla kirjoitettua konetekstiä. Toisaalta esimerkkiverkko suhtautuu erittäin huonosti inhimilliseen vaihteluun syötteissä. Sillä olisi ongelmia tunnistaa oikein numero yksi, joka on kirjoitettu syötekentän vasempaan laitaan. Syötteiden normalisointi, esimerkiksi keskittäminen ja kontrastin säätö, ratkaisisivat ongelma osittain, mutta on vaikea toteuttaa luotettavasti [LeCun *et al.* 1998a].

Tätä ongelmaa voidaan kutsua paikalliseksi riippuvuudeksi. Luokittelun onnistuminen on riippuvainen siitä, missä kohtaa syötevektoria, eli missä päin kuvaa, luokiteltava kohde sijaitsee. Rakenteettomat verkot, joissa kaikki neuronit ovat yhdistettyinä toisiinsa, voivat teoriassa oppia luokittelemaan paikallisesti riippumattomasti, mutta niiden koulutus on käytännössä erittäin vaikeaa [LeCun *et al.* 1998a]. Tällainen verkko tarvitsisi erittäin paljon harjoitusesimerkkejä miljoonien painojen opettamiseksi.

Fukushiman ”neocognitron” [1980] sisältää kaksi perusmenetelmää paikallisen riippumattomuuden saavuttamiseksi. Ensinnäkin syötteelle olennaisia piirteitä kannattaa etsiä joka puolelta syötettä. Esimerkiksi tietynlaista kaarta voidaan ja kannattaa etsiä eri puolilta käsinkirjoitettua numeroa. Tämä toteutetaan tietynlaisilla suodattavilla kerroksilla verkoissa, jotka etsivät jotain samaa piirrettä koko edellisen kerroksen alueelta.

Toiseksi löytyneen piirteen tarkka sijainti ei useimmiten ole kuvantunnistuksessa merkittävää luokittelun kannalta. Järkevämpää on tutkia, löytyykö tiettyä piirrettä joltain tietyltä summittaiselta kerroksen alueelta. Summittaisuus saavutetaan ryhmittämällä piirre-etsijöiden antamia aktiviteettejä paikallisiin naapurustoihin. Näistä naapurustoista voidaan tutkia aktiviteettien keskiarvoa tai maksimia.

Havainnollistan näitä menetelmiä kuvassa 5. Tavanomaisten, täysin yhdistettyjen kerrosten sijaan suodattavissa verkoissa on suodattavia (*convolutional*) ja tiivistäviä (*maxpooling, average pooling)* kerroksia. Erilaisten piirteiden etsintä voidaan nähdä konvoluutiona, jossa suodattimena toimiva matriisi liukuu syötekuvan päällä [LeCun *et al.* 1998a]. Tällainen T-tyyppisiä piirteitä etsivä matriisi nähdään esimerkissä. Konvoluution tulos muodostaa suodattavan kerroksen, jota voidaan myös kutsua piirrekartaksi (*feature map*). Valmis piirrekartta kuvaa siis, missä kohdissa syötettä suodatin aktivoituu. Esimerkissä suodatinta ei käytetä siten, että osa siitä olisi syötteen reunojen ulkopuolella. Täten piirrekartan koko on välttämättä pienempi kuin syötteen.



Kuva 5. Karkea esimerkki suodattavan verkon paikallisesta riippumattomuudesta.

Kun piirrekartta on laskettu, tiivistetään siitä tietynkokoisen neuroneiden naapurusto. Esimerkissä tiivistäminen tapahtuu etsimällä 5x5 kokoisen neuroninaapuruston maksimiaktiviteetti. Kuvasta nähdään, miten kaksi erilaista syötettä tuottavat samanlaisen representaation tiivistävään kerrokseen.

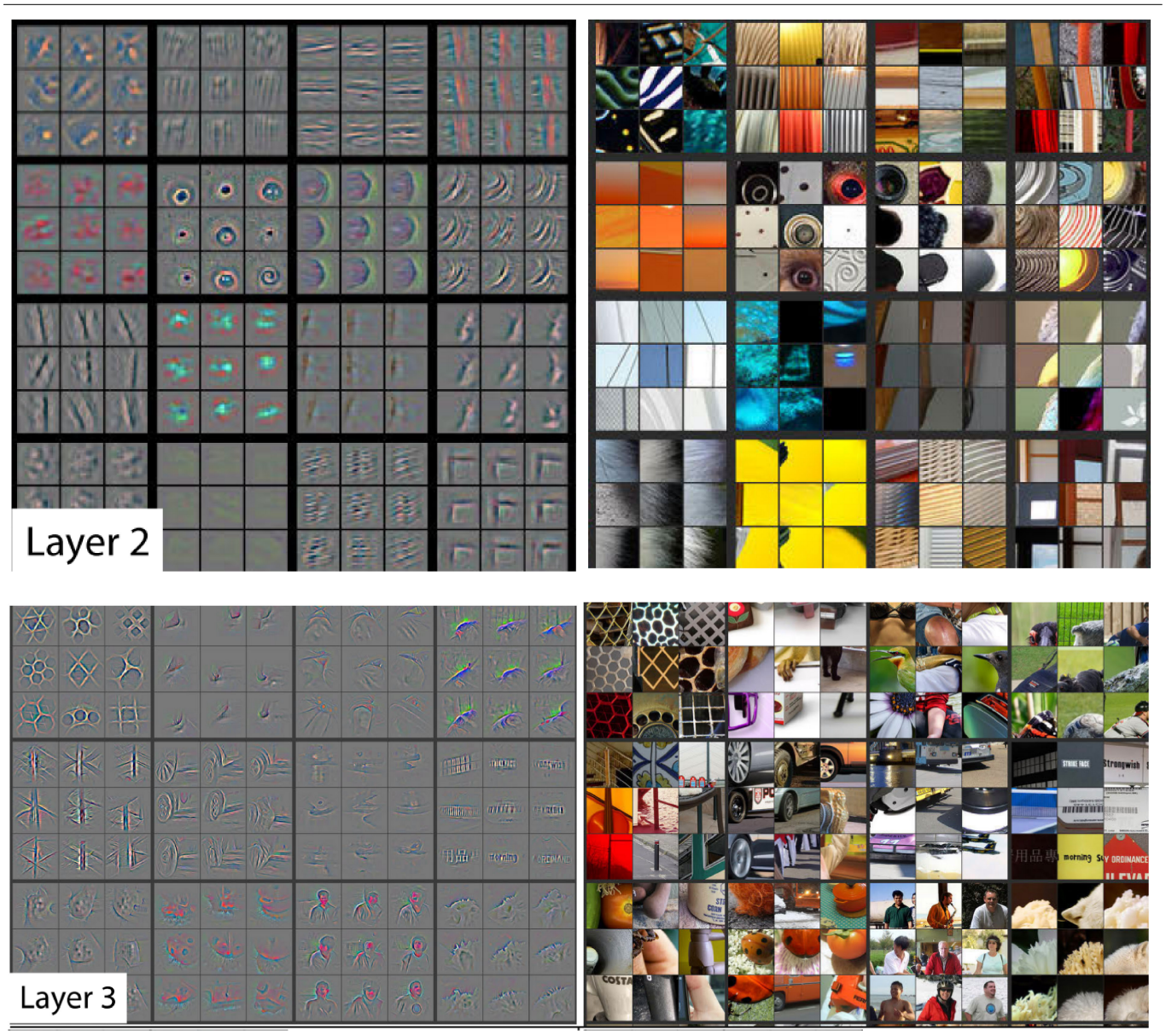
Ketjuttamalla suodatus-tiivistys -kerrospareja syötevektori saadaan muunnettua joukoksi erilaisia pieniulotteisempia representaatiota. Verkon viimeinen tiivistävä kerros on usein täysin yhdistetty yksiulotteiseen kerrokseen, joka on täysin yhdistetty tuloskerrokseen. Esimerkiksi vuonna 1998 toteutetussa LeNet5:ssä [LeCun *et al.* 1998a] suodatus-tiivistys -kerrospareja on kaksi, joiden jälkeen verkossa on kaksi täysin yhdistettyä kerrosta ja tuloskerros.

Suodattavia matriiseja kerrosten välillä on aina enemmän kuin yksi. LeNet5:ssä syötekerros suodatetaan ensimmäiseen suodattavaan kerrokseen kuudeksi erilaiseksi piirrekartaksi eli suodattimia on käytössä kuusi. Tuoreemmassa liikennemerkkejä luokittelevassa neuroverkossa [Cireşan *et al.* 2012], piirrekarttoja on ensimmäisessä suodattavassa kerroksessa 100 kappaletta ja kolmannessa 250.

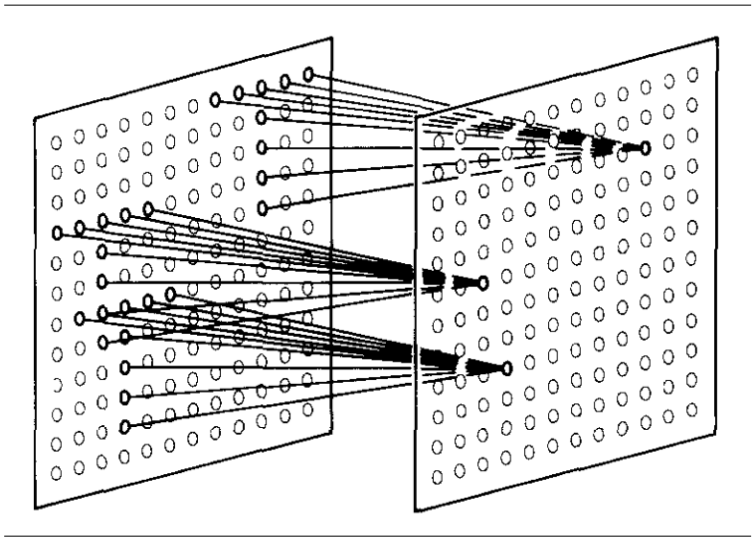
Ensimmäisen kerrosparin jälkeen suodattavat matriisit etsivät piirteitä useammasta tiivistävästä kartasta samaan aikaan. Täten syvemmälle verkossa edetessä matriisien etsimien piirteiden kompleksisuus kasvaa. Syvempien kerrosten matriisit voidaan nähdä erilaisten yksinkertaisten piirteiden yhdistelmien etsijöinä. Ilmiötä havainnollistavat

Zeiler ja Fergus [2013] näyttämällä millaisia aktivaatioita eri suodattimet saavat tietynlaisilla kuvilla. Kuva 6 esittää satunnaisesti valittujen piirresuodattimien aktivoitumista eri tasoilla. Kuvasta nähdään, miten syvempien kerrosten piirresuodattimet havainnoivat monimutkaisempia piirteitä kuin lähellä syötettä olevat.

Käytännössä suodattavat kerrokset toteutetaan jaetuilla painoilla. Syötekerroksesta lähtee tietynlainen toistuvien painojen kuvio jokaiseen seuraavan kerroksen neuroniin. Tämän mallin Fukushima [1980] havainnollistaa hyvin. Kuvasta 7 näkee, miten T-kuviota kuvaavat painot toistuvat jokaiseen suodattavan kerroksen neuroniin. Tässä esimerkissä voidaan kuvitella, että ainoastaan neuronia virittävät painot on piirretty näkyviin. Neuronin aktivoituminen tällöin kuvaa sitä, esiintyykö tietynlaista kuviota edellisellä tasolla tietyssä kohdassa.



Kuva 6. Satunnaisesti valittujen piirresuodattimien aktiviteetit kahdessa eri kerroksessa. Vasemmalla on suodattimen aktiviteetit ja oikealla yhdeksän kuvan osaa, jotka aiheuttivat korkeimman kokonaisaktiivisuuden [Zeiler and Fergus 2013].



Kuva 7. Suodatus toteutetaan käyttämällä jaettuja painoja yhteyksissä edelliseen kerrokseen [Fukushima 1980].

Suodattavilla ja tiivistävillä kerroksilla saavutetaan jonkinasteinen riippumattomuus piirteen sijainnista syötealueella. Toinen, jaettujen painojen, tuoma etu on opittavien parametrien määrän pienuus. Esimerkiksi LeNet5 [LeCun *et al.* 1998a] sisältää 345 308 yhteyttä, mutta vain 60 000 opeteltavaa parametriä. Näytönohjainten kehityksen myötä kasvanut laskuteho on mahdollistanut syvemmät ja leveämmät verkot, joita laskenta- ja muistivaatimukset kuitenkin edelleen rajoittavat. Suurehkon verkon kouluttamiseen kuluu edelleen monta päivää [Krizhevsky *et al.* 2012].

3.2 Monen verkon komiteat

Cireşan ja muut [2011; 2012] esittävät neuroverkkojen kuvantunnistustarkkuuden parantamiseksi monen suodattavan verkon yhdistämistä komiteaksi. Idean takana on havainto siitä, että neuroverkoilla on tapana tehdä erilaisia virheitä luokittelutehtävissä. Toisin sanoen virheiden määrää saadaan vähennettyä keräämällä yhteen monen eri neuroverkon arvaus luokasta.

Komiteaan kuuluvat verkot koulutetaan toisistaan hieman eroavilla harjoitussyötteillä. Aikaisemmissa kokeissaan Cireşan ja muut [2011] käyttävät originaalin harjoituskokoelman lisäksi kuudelle eri leveydelle venytettyjä tai kavennettuja harjoituskuvia. Kullekin harjoituskokoelmalle koulutetaan viisi verkkoa, jolloin verkkoja on yhteensä 35 kappaletta.

Yksittäisten verkkojen tulokset paranevat huomattavasti, kun eri harjoituskokoelmilla oppineista verkoista muodostetaan seitsemän verkon komiteoita. Käsinkirjoitetuissa merkeissä yksittäisen verkon virheprosentti on noin 14%, kun taas erilaisten seitsemän verkon komiteoiden keskivirhe on hieman alle 12% [Cireşan *et al.* 2011]. Mielenkiintoinen havainto on myös se, että painojen satunnaisen alustamisen vuoksi kahdella täysin samalla harjoituskokoelmalla ja oppimismenetelmällä oppineella verkolla saattaa olla prosentinkin ero tunnistusvirheessä. Samassa käsinkirjoitettujen merkkien

tunnistustehtävässä kahden lähtökohdiltaan samanlaisen verkon virheprosentit ovat 14.7% ja 13.7% [Cireşan *et al.* 2011].

Monen verkon komiteat saavuttavat yksittäisiä verkkoja paremmat tulokset monissa merkkien ja kuvien luokittelutehtävissä. Suurin haittapuoli monen verkon käytössä on koulutuksen vaatima aika. Esimerkiksi liikennemerkkejä luokittelevan 25 verkon ryhmän koulutus kestää neljällä näytönohjaimella 37 tuntia [Cireşan *et al.* 2012]. Jos verkko on leveämpi ja syvempi, ja yhden verkon oppimiseen kuluu monta päivää, on kymmenien verkkojen kouluttaminen samalla laitteistolla epäkäytännöllistä. Oppineen verkkokomitean toiminta sinänsä ei vaadi rinnakkaisia näytönohjaimia toimiakseen riittävällä nopeudella, koska valmiin syötteen tutkiminen on operaationa paljon oppimista kevyempi.

3.3 Suodattavat verkot toiminnassa

Neuroverkkoja ollaan kokeiltu moniin kuva-analyysin sovelluksiin. Luokittelutehtävissä on vaikeuseroja, jotka vaikuttavat luokitteluvirheen määrään ja käytettyjen verkon rakenteeseen. Syötteiden normalisointi, opetuksen tukeminen ja verkon asema kokonaisjärjestelmässä ovat myös vaihdelleet.

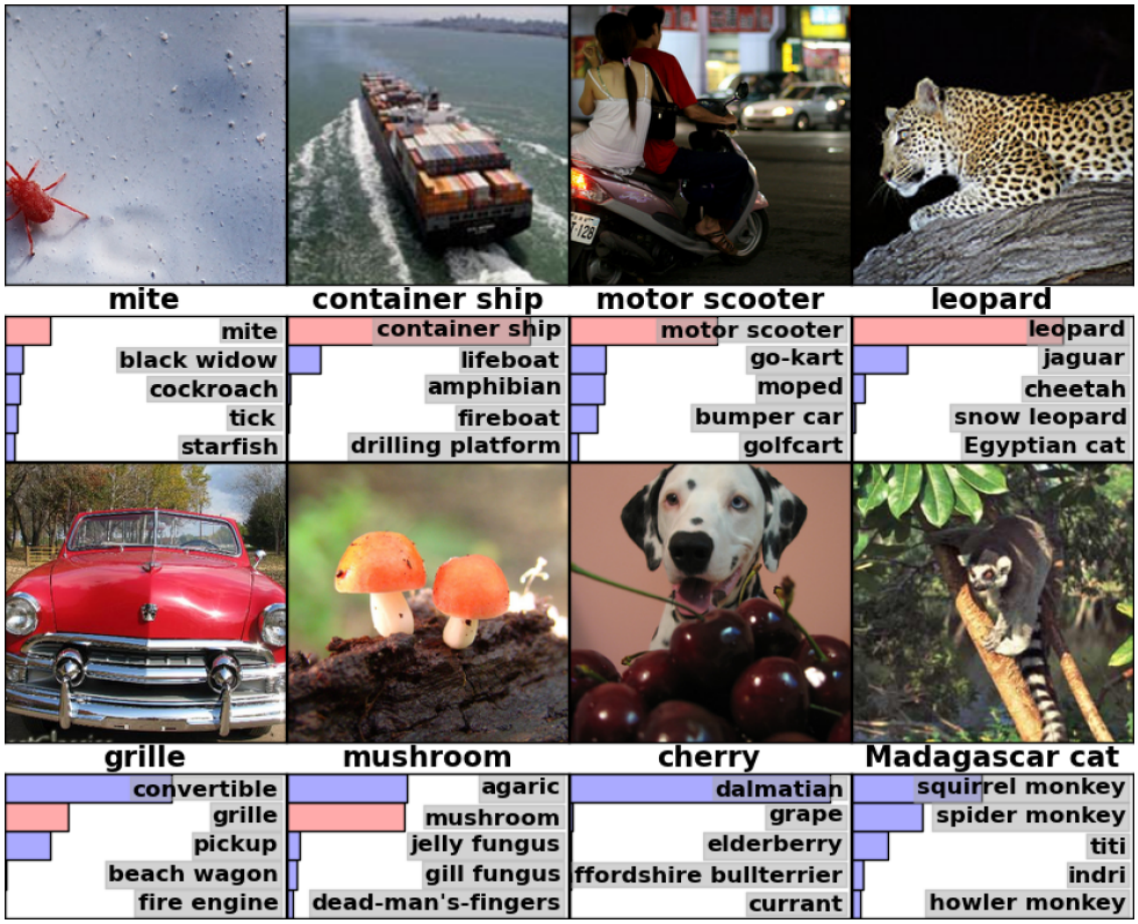
Käsinkirjoitetun numeron luokitteluun suodattavia verkkoja ollaan käytetty jo 1980-luvulta alkaen [LeCun *et al.* 1989]. Luokitteluvirhettä on nykyisillä verkoilla parhaimmillaan 0.23% [Cireşan *et al.* 2012]. Tehtävän tekee helpoksi se, että kohdeluokkia on vähän ja saman luokan syötteiden välillä suhteellisen pientä vaihtelua. Vaikeampi tehtävä on luokitella latinalaisen aakkoston kirjaimia tai kiinalaisia merkkejä. Näissä tehtävissä saavutetaan 7.4% ja 6.5% virhetaso [Cireşan *et al.* 2012]. Hyviä luokittelutuloksia selittää osittain se, että neuroverkkojen kehityksessä käytettävät merkkitietokannat ovat kooltaan ja kontrastiltaan lähes normalisoituja, mikä lähentää samaan luokkaan kuuluvia syötevektoreita.

Lähellä merkintunnistusta on tekstuurin luokittelu. Samoin kuin merkit on tekstuuri luontaisesti litteä. Puulajien mikroskooppisten poikkileikkauskuvien tunnistuksessa saavutetaan pienempi virheprosentti, 2.7%, kuin merkistöjen tunnistuksessa [Hafemann *et al.* 2014]. Oletettavasti eri puuyksilöiden solurakenteissa on vähemmän yksilöiden välistä variaatiota kuin esimerkiksi eri käsialojen välillä. Toinen esimerkki tekstuurin luokittelusta on värjättyjen solunäytteiden analyysi [Cireşan *et al.* 2013].

Luokittelu vaikeutuu olennaisesti, jos tunnistettavat luokat ovat esineitä tai olioita. Samasta kolmiulotteisesta esineestä voidaan ottaa rajaton määrä erilaisia kaksiulotteisia projektioita, joten samaan luokkaan kuuluvilla kuvilla on enemmän keskinäistä vaihtelua. Luokan kannalta oleellinen tieto ei myöskään sijaitse aina tietyn kokoisena, tietyssä kohdassa syötettä. Verrattuna merkkeihin, syötteen kannalta olennaista on ottaa mukaan kaikki värit, mikä laajentaa verkon rakennetta. Esine- ja oliokuvien luokittelu on siis luontaisesti merkkiluokittelua vaikeampi tehtävä.

Luonnollisten kuvien luokittelussa voidaan mitata erikseen ns. top-1- ja top-5 -virheitä. Top-1 -virhe kertoo, kuinka usein neuroverkon ensimmäinen arvaus tekee vir-

heitä ja top-5 ottaa huomioon viisi ensimmäistä arvausta. Top-5 -virhe on hyödyllinen mittari silloin, kun kohdeluokkia on paljon. Tällainen luokittelutehtävä on esimerkiksi ILSVRC2012-kilpailu, jossa luonnollisia valokuvia luokitellaan tuhanteen eri luokkaan. Krizhevsky ja muut [2012] saavuttavat erilaisilla suodattavilla verkoilla ja niiden komiteoilla top-1 -virheeksi noin 38% ja top-5 -virheeksi 16%. Virheet ovat suuria verrattuna merkintunnistukseen, mutta esimerkiksi top-5 -virheessä kehitys edelliseen parhaaseen tulokseen on noin 10%. Verkon rakenne on massiivinen: opittavia parametrejä on 60 miljoonaa ja oppiminen kestää kuusi päivää.

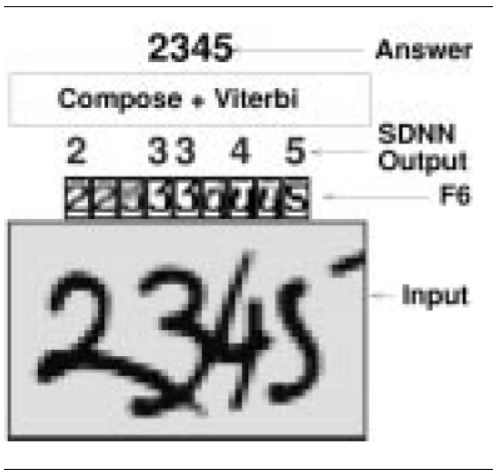


Kuva 8. ImageNet-tietokannan kuvia ja neuroverkon luokitteluarvauksia. Heti kuvan alapuolella lukee kuvan oikea luokka, ja tämän alla on neuroverkon ensimmäiset viisi arvausta luokaksi. Palkin leveys kuvaa arvauksen varmuutta [Krizhevksy *et al.* 2012].

Kuvassa 8 on esimerkkejä yllä mainitun verkon tuloksista. Esimerkeistä nähdään, miten syötteet eivät ole segmentoituja tai normalisoituja: tunnistettava esine tai olio saattaa sijaita missä tahansa kohdassa syötettä vaihtelevan kokoisena. Luokittelua helpottaa, jos oleellinen syöte saadaan jotenkin leikattua irti kuvan epäolennaisesta taustahälystä eli segmentoitua. Oleellisen osan etsimiseen voidaan käyttää erilaisia heuristisia

menetelmiä, jotka hyödyntävät tietoa sovellusalueesta. Esimerkiksi liikennemerkkejä luokitteleva verkko etsii koko syötealueesta tiettyjä värejä, ja löydetyistä värialueesta tiettyjä muotoja [Broggi *et al.* 2007]. Tällöin neuroverkon tehtäväksi jää ainoastaan tarkasti rajatun syötealueen luokittelu. Syötettä voidaan myös normalisoida, mikä tarkoittaa esimerkiksi kontrastin parantamista tai tiettyyn kokoon suurentamista.

Oleellisen osan segmentointi ja normalisointi ei ole helppoa. Esimerkiksi kaunokirjoitettuja kirjaimia on hankala erotella toisistaan omiksi syötteikseen. Neuroverkoilla ongelmaa voidaan kiertää tutkimalla usealla vierekkäisellä verkolla koko syötealuetta, jolla oleellinen syöte sijaitsee [LeCun *et al.* 1998a]. Kukin suodattava verkko tuottaa arvauksen omasta syötealueestaan. Koska syötealueet ovat limittäin, vierekkäiset verkot saattavat tunnistaa oikein saman merkin (kuva 9). Neuroverkkojen lisäksi järjestelmä vaatii osan, joka tulkitsee tulosvektoreiden sarjasta parhaan mahdollisen yhdistelmän. Yllä mainittu järjestelmä tutkii painotettujen graafien avulla, mikä yhdistelmä saa pienemmän mahdollisen sakon ja hyväksyy sen vastaukseksi. Limittäisten verkkojen määrä ja tulosten tulkinta luonnollisesti lisää kokonaisjärjestelmän laskennallista hintaa.



Kuva 9. Syötekenttää tutkii monta verkkoa, joiden syötealueet ovat hiukan limittäin. Kuvan syötteestä kaksi vierekkäistä verkkoa ovat molemmat tunnistaneet numeron 3 [LeCun *et al.* 1998a].

Vaikeissa tehtävissä neuroverkoilla ei vielä saavuteta ihmismäistä tarkkuutta, vaikka käsittelynopeudessa ihminen ylitetään moninkertaisesti. Neuroverkko voi olla vaikeissa kohteissa inhimillisten päätösten tukena suodattamassa massiivisia määriä dataa ja kiinnittämässä käyttäjän huomion datan olennaisiin osiin. Verkon koko tulostason aktiivisuutta tarkastelemalla voidaan arvioida arvauksen varmuutta. Matalan varmuuden tapauksissa järjestelmä voi turvautua ihmisen apuun. Enemmän vastuuta verkolle voidaan antaa järjestelmissä, jossa virheet eivät aiheuta käyttäjilleen kriittisiä seurauksia.

4. Koneen opettaminen näkemään

4.1 Oppiminen korvaa käsinkoodatut piirteet

Koneoppimisen yhtenä tavoitteena on ratkaista ongelmia, joita emme pysty tarkasti formalisoimaan. Dahl [2015] toteaa koneoppimisen tarjoavan meille uuden näkökulman ohjelmointiin: voimme kertoa millaiseen lopputulokseen algoritmin pitäisi päätyä, mutta ei ole välttämätöntä kertoa, miten tulokseen päädytään. Neuroverkkojen oppiminen voidaan nähdä vastaavana prosessina, jossa verkolle kerrotaan luokittelufunktion oikea vastaus, ja verkon tehtäväksi jää luokittelufunktion yksityiskohtien arvioiminen.

Tämä on ehkä neuroverkon tärkein ominaisuus vaikeiden tunnistustehtävien kannalta. Luokitteluun käytettäviä piirretunnistajia ei tarvitse koodata käsin, ja luokittelija voidaan päivittää uuteen luokiteltavaan sisältöön uusilla harjoitusesimerkeillä. Toisaalta tehtävään liittyvää tietoa pitää hyödyntää neuroverkon rakenteen suunnittelussa. Suodattavissa verkoissa piirteitä etsitään paikallista naapurustoista eri puolelta syötettä. Tämä rakenteellinen ratkaisu ottaa huomioon kuvan kaksiulotteisen erikoislaadun syötteenä. Läheisten pikseleiden muodostamat kokonaisuudet ovat tärkeämpiä kuin esimerkiksi syötekentän eri nurkissa sijaitsevien yhteys.

4.2 Peruuttaminen funktion pohjalle – matemaattiset perusteet

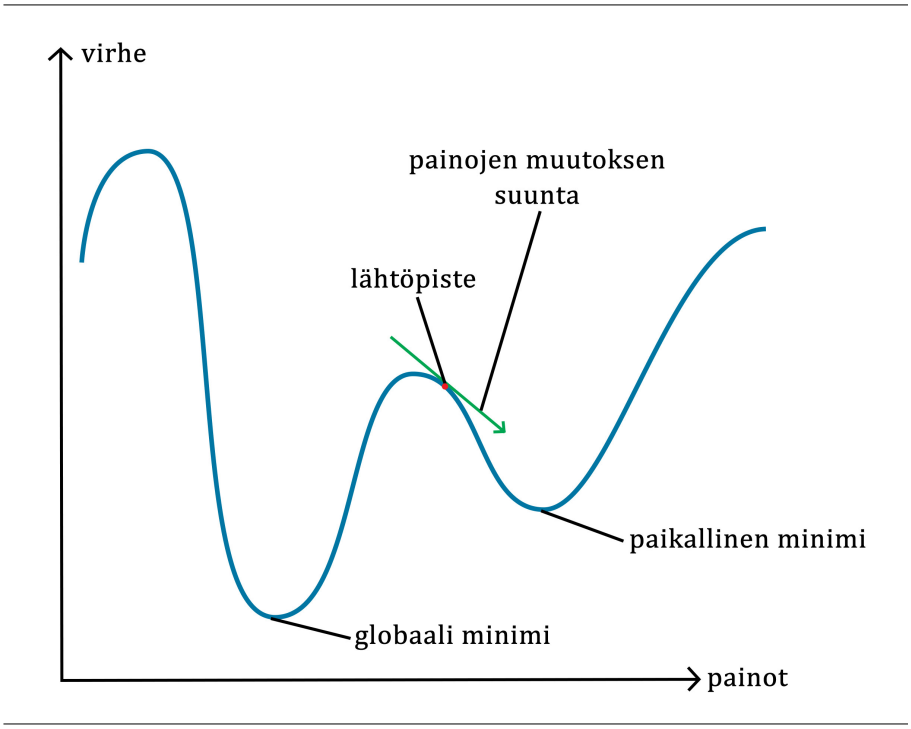
LeCunia ja muita [1998b] mukaillen neuroverkko voidaan nähdä funktiona $F(Z^p, W)$, jossa Z^p on syöte järjestysnumerolla p ja W verkon sisältämät painot. Luokittelutehtävässä tämän funktio tulos voisi olla luokka-arvauksia kuvaava vektori. Kullakin syötteellä virhettä kuvaa funktio $E^p = C(T^p, F(Z^p, W))$, jossa funktio C laskee virheen kohdetuloksen ja funktiosta F saadun tuloksen välillä. Tavoitteena on minimoida virhe kaikilla syötteillä.

Käytännössä virhe minimoidaan etsimällä mahdollisimman pienen virheen tuottavat painot. Pysähdytään käsittelemään yksittäistä harjoitussyötettä. Tulostason neuronin aktiivisuus, ja tätä kautta myös virhe, on tässä tilanteessa jokin edellisen kerroksen aktiivisuuden ja painojen funktio. Kun edellisen kerroksen aktiivisuus on vakio, voidaan painoja ajatella tämän funktion muuttujina. Tilannetta vastaa avaruus, jossa vaakataso kuvaa erilaisia painojen yhdistelmiä ja pystytaso virheen määrää [Hinton 2012].

Virhe laskettuna eri painoilla muodostaa virhepinnan tähän avaruuteen ja nykyiset painot ovat yksi piste virhepinnalla. Tässä pisteessä voidaan virheelle laskea osittainen derivaatta jokaisen painon suhteen eli *gradientti*. Tuloksena on vektori, joka osoittaa virheen jyrkimmän nousun suunnan. Kun painoja muutetaan pieni määrä tämän jyrkimmän nousun vastakkaiseen suuntaan, pitäisi virheen pienentyä.

Kuvassa 10 on esimerkki virhepinnasta ja painojen muutoksesta. Kuva havainnollistaa myös ongelmaa, joka tällä menetelmällä teoriassa on. Virhepinta on muodoltaan monimutkainen, eikä jossain tietyssä kohtaa laskettu gradientin vastasuunta välttämättä johda suoraan globaalisti pienintä virhettä kohti. LeCunin ja muiden [1998a] mukaan paikalliset minimit eivät yleensä aiheuta neuroverkoille ongelmia, mutta tarkkaa teo-

reettista selitystä tähän ei tiedetä. Yksi menetelmä lokaalisten minimien välttämiselle on useamman verkon opettaminen rinnakkain [Atakulreka and Sutivong 2007]. Satunnaisten alustuksen vuoksi painojen pitäisi kehittyä kohti eri virheminimeitä. Opetuksen edetessä eniten virhettä tuottavat verkot voidaan pudottaa pois, jolloin menetelmän laskennallinen raskaus pysyy kohtuullisena.



Kuva 10. Painot saattavat siirtyä lähtöpisteestä kohti paikallista minimiä eivätkä välttämättä ikinä päädy globaaliin minimiin.

Rumelhart ja muut [1986] esittävät ja todistavat toimivaksi menetelmän, joka toteuttaa tällaisen laskeutumisen virheenfunktion pohjalle. Tätä menetelmää kutsutaan peruutusalgoritmiksi (*backpropagation algorithm)*. Nimi perustuu tapaan, jolla painomuutokset lasketaan kussakin kerroksessa tuloskerroksesta verkon alkupäähän peruuttaen. Menetelmä voidaan jakaa kahteen vaiheeseen [Rumelhart *et al*. 1986]:

1) Neuroverkko käsittelee syötteen ja tuottaa jonkin tulosarvon jokaiselle tuloskerroksen yksikölle, ja jokaiselle yksilölle lasketaan virhesignaali vertailemalla tuloksia kohdetulokseen.
2) Virhesignaali lasketaan jokaiselle kerrokselle rekursiivisesti viimeisestä kerroksesta lähtien aina yksi kerros taaksepäin peruuttaen.

Yhden painon muutos missä tahansa kohtaa verkkoa on

$$\Delta_p w_{ij} = \eta \delta_{pj} o_{pi}. \qquad (2)$$

Muutettava paino on w_{ij}, ja se sijaitsee neuroneiden *i* ja *j* välissä. Muutettava määrä on oppimiskerroin η kertaa neuronin *j* virhesignaali δ_{pj} kertaa neuronin *i* aktiviteetti o_{pi}. Indeksi *p* muistuttaa, että kyseessä on muutokset tietyn harjoitusesimerkin *p* kohdalla. Oppimiskerroin η on sopivasti valittu vakiokerroin, joka pitää huolta, että painojen muutokset ovat sopivan kokoisia. Oppimiskertoimen merkitykseen palataan alla.

Virhesignaali δ_{pj} riippuu siitä, sijaitseeko neuroni *j* tuloskerroksessa vai verkon keskellä. Peruutusalgoritmissa virhesignaalit pitää laskea ensiksi tuloskerrokselle. Tuloskerroksen virhesignaali on

$$\delta_{pj} = (t_{pj} - o_{pj}) f'_j(net_{pj}), \tag{3}$$

jossa $(t_{pj} - o_{pj})$ on tavoitetuloksen ja saadun tuloksen erotus, $f'_j(net_{pj})$ aktivointifunktion derivaatta ja net_{pj} neuroniin tullut aktiviteettien ja painojen tulojen summa (ks. kaava 1). Jos lasketaan piilotetun kerroksen neuronin virhesignaalia, on kaavana

$$\delta_{pj} = f'_j(net_{pj}) \sum_k \delta_{pk} w_{kj}. \tag{4}$$

Kaavan etuosa on sama derivaatta kuin kaavassa 3. Jälkiosassa summataan seuraavan kerroksen virhesignaalit kerrottuna seuraavaan kerrokseen menevillä painoilla. Tavallaan jälkiosa vastaa verkon normaalia toimintaa, mutta nyt aktiviteetin tilalla on virhesignaalit ja verkossa edetään peruuttaen.

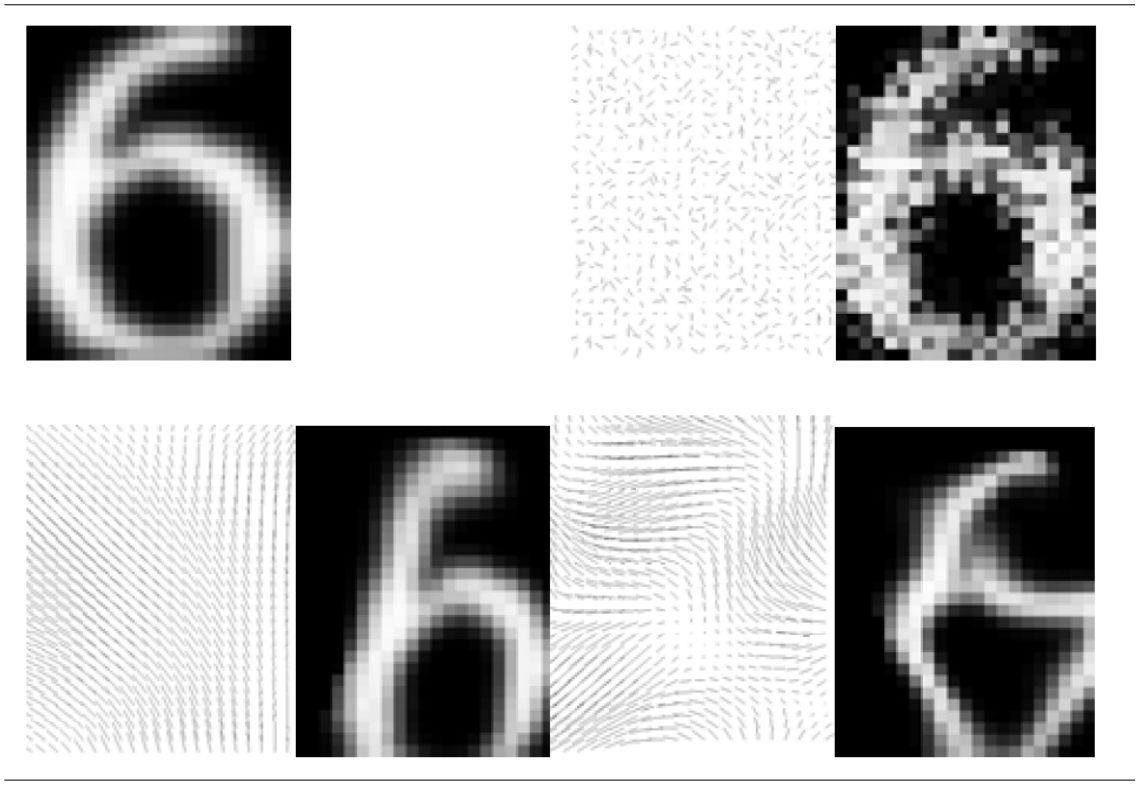
Yksittäisen painon muutoksen kaavassa esiintyy oppimiskerroin η. Oppimiskerroin määrittää, minkä kokoisia muutoksia painoihin tehdään. Gradientin vastasuunnasta tiedetään ainoastaan, että se osoittaa virhettä pienentävään suuntaan *paikallisesti*, mutta ei välttämättä suoraan kohti virhefunktion pohjaa. Toisaalta emme tiedä, kuinka kaukana virhefunktion pohja on. Oppimiskerroin η on valittava siten, että painot eivät päädy heiluriliikkeeseen, mutta toisaalta oppimista tapahtuu riittävällä nopeudella [Rumelhart *et al.* 1986]. Oppimiskerrointa kannattaa pienentää oppimisen kuluessa, kun painot alkavat olla lähäempänä minimivirheen tuottamista [LeCun *et al.* 1998a].

Painoja voidaan muuttaa joko jokaisen harjoitustapauksen tai harjoitustapausjoukon läpikäymisen jälkeen. Jatkuvassa painojen muuttamisessa (*stochastic gradient descent*) on tiettyjä etuja [LeCun *et al.* 1998b]. Yhtäältä jatkuva muuttaminen johtaa nopeampaan virheen minimointiin. Toisaalta painomuutoksissa on useimmiten enemmän hälyä kuin harvemmissa muutoksissa. Toisin sanoen painot hyppelevät satunnaisemmin eri suuntiin painoavaruudessa. Tämä johtaa siihen, että paikallisiin minimeihin ei jäädä jumiin. Jatkuva muuttaminen ei ole yksiselitteisesti parempi menetelmä, ja se painot saattavat heilahtelemaan paikallisen minimin ympärille [LeCun *et al.* 1998b]. Suurin osa nykyaikaisista kuvantunnistuksen neuroverkoista käyttää jatkuvaa painojen säätöä.

4.3 Ylisopeutuminen välttäminen

Kuvantunnistuksessa neuroverkon oppimisen tavoitteena on paitsi minimoida harjoitusvirhe myös yleistyä luokittelemaan harjoitteluhetkellä tuntemattomia syötteitä. Yleistyminen tarkoittaa esimerkiksi sitä, että eri kuvakulmista ja etäisyyksillä otetut valokuvat samasta esineestä osataan luokitella samoiksi. Verkko ei saa ylisopeutua (*overfit*) harjoitusdataan.

Suurilla verkoilla ylisopeutumisen välttäminen vaatii massiivisia määriä harjoitteludataa, joka kuvaa mahdollisimman hyvin kaikenlaisia syötteitä, joita valmis neuroverkko joutuu luokittelemaan. Yksi keino välttää ylisopeutumista on uusien harjoitussyötteiden luominen. Olemassaolevia syötteitä transformoimalla saadaan luotua uusia [Simard *et al*. 2003]. Merkkien tunnistuksessa luontimenetelmiä ovat esimerkiksi kuvan venyttäminen [Simard *et al.* 2003; Cireşan *et al.* 2012] ja kohinan tai sotkun lisääminen [Simard *et al.* 2003]. Kuvassa 11 nähdään esimerkit molemmista menetelmistä.



Kuva 11. Vasemmalla ylhäällä alkuperäinen harjoitussyöte, oikealla ylhäällä ja alhaalla erilaisia transformaatioita [Simard *et al.* 2003].

Esineiden ja olioiden luokittelussa Krizhevsky ja muut [2012] leikkaavat 256x256 pikselin kuvasta satunnaisia 224x224 pikselin kokoisia alueita ja näiden horisontaalisia peilikuvia harjoitussyötteiksi. Tämä on suodattavan verkon rakenteen kanssa toimiva lisäkeino paikallisen riippumattomuuden saavuttamiseksi. Toinen Krizhevskyn ja muiden [2012] käyttämä menetelmä on pikselien väriarvojen muuntelu. Jokaisesta harjoitus-

syötteestä saadaan täten useampi syöte, jotka kuvaavat luokiteltavan kohteen yleisemmin kuin yksittäinen valokuva.

Hiljattain keksitty keino ylisopeutumisen välttämiseksi on neuroneiden sammuttaminen (*dropout*) harjoituksen aikana [Hinton *et al.* 2012]. Sammuttamisessa noin puolet piilotettujen, täysin yhdistettyjen kerrosten neuroneista suljetaan pois käytöstä kunkin harjoitussyötteen kohdalla. Menetelmän pitäisi vähentää verkon sopeutumista testidataan tekemällä neuroneista vähemmän riippuvaisia tiettyjen toisten neuroneiden toiminnasta. Myös syöteyksiköistä voidaan osa sulkea oppimisen ajaksi pois päältä. Tavallaan jokaisesta syötteestä osa piilotetaan. Yhdestä syötteestä esitetään täten monta eri representaatiota verkolle, minkä pitäisi vähentää verkon sopeutumista yhteen tiettyyn syötteeseen. Hinton ja muut [2012] kokeilevat sammuttamista useaan kuvanluokittelutehtävään, joista jokaisessa luokitteluvirhe pienenee merkittävästi.

4.4 Nopea katsaus aktivointifunktioihin

Neuroverkoissa on yleisimmin käytetty aktivointifunktioina tanh-funktiota [LeCun *et al.* 1998a] ja sigmoidifunktioita [Rumelhart *et al.* 1986]. Mainituilla funktioilla neuronin lähettämä aktiviteetti saadaan rajoitettuja tiettyjen reaaliarvojen, kuten 0 ja 1 välille, mutta muutos arvojen välillä on jatkuvaa. Sigmoidin kaava on

$$f(x)=\frac{1}{1+e^{-x}} \tag{5}$$

ja tanh-funktion

$$f(x)=\frac{1-e^{-2x}}{1+e^{-2x}}. \tag{6}$$

Tuore tutkimus [Glorot *et al.* 2011; Krizhevsky *et al*. 2012] on osoittanut ns. saranafunktion (*rectifier function*) tai softplus-funktion parantavan oppimista ja verkon aktiviteettien niukkuutta (*sparsity*). Saranfunktio on yksinkertaisesti

$$f(x)=max(0,x) \tag{7}$$

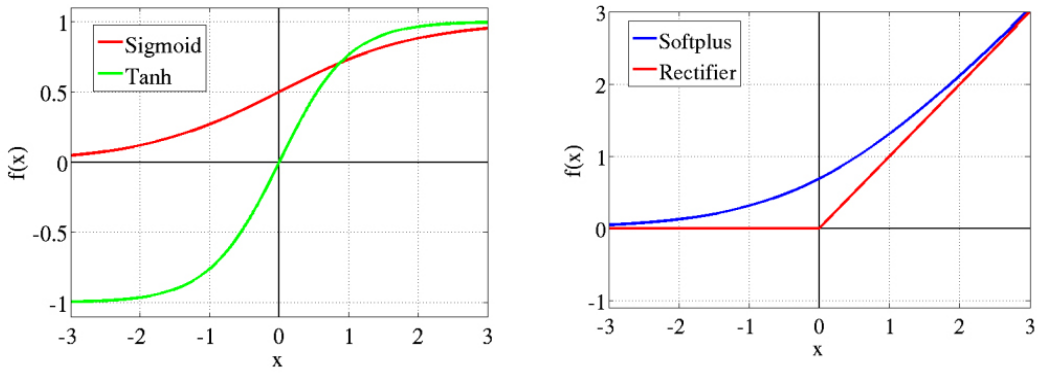
ja softplus

$$f(x)=\ln(1+e^{x}). \tag{8}$$

Kaikkien funktioiden kuvaajat löytyvät kuvasta 12**.**

Aktivointifunktion pitää olla melko yksinkertainen, sillä aktivointeja tapahtuu yhden syötteen käsittelyssä satoja tuhansia. Toinen vaatimus on jatkuva derivoituvuus, joka johtuu virhesignaalien laskutavoista (kaavat 3 ja 4). Sigmoidin ja tanh-funktion etu on jatkuva muutos, jolloin derivaatta on aina enemmän kuin nolla. Derivaatta tosin muuttuu häviävän pieneksi erittäin suurilla ja pienillä arvoilla. Saranafunktion kuvaajasta nähdään, että sen derivaatta on 0, kun $x \leq 0$. Derivaatalla nolla oppimista ei ta-

pahdu, joten saranafunktiota käyttävän neuronin pitää saada ainakin joitakin positiivisia syötesummia oppiakseen [Krizhevsky *et al.* 2012]. Softplus-funktio toimii käytännössä jatkuvasti arvoaan muuttavana versiona saranafunktiosta.



Kuva 12. Aktivointifunktioiden kuvaajia [Glorot *et al.* 2011].

Saranafunktion käyttöä perustellaan sen tuottamalla aktiviteettien niukkuudella verkossa. Niukkuus tarkoittaa sitä, että aktiviteetti on suurimmalla osalla neuroneista tasan nolla verkon käsitellessä syötettä. Niukasti aktivoituvalla verkolla on tiettyjä etuja [Glorot *et al.* 2011]. Saranafunktio on muun muassa vähemmän herkkä pienille muutoksille syötteissä verrattuna jatkuvasti arvoaan muuttaviin funktioihin. Ylipäätään niukasti aktivoituva verkko saa helpommin toisistaan erotettavia tiloja erilaisilla syötteillä. Saranafunktio on myös biologisesti uskottavampi kuin tanh- tai sigmoidifunktio [Glorot *et al.* 2011]. Käytännössä saranafunktiota käyttävä verkko oppii nopeammin kuin tanh-funktiota käyttävä [Krizhevsky *et al.* 2012].

5. Yhteenveto

Suodattavia neuroverkkoja on onnistuttu käyttämään vaihtelevalla menestyksellä erilaisiin kuvantunnistustehtäviin. Käsinkirjoitettujen numeroiden lukemisesta suoriudutaan lähes ihmisen tarkkuudella ja käsinkirjoitettujen merkkien luvusta melko hyvin. Vaikeinta on tunnistaa kolmiulotteisista olioista ja esineistä otettuja kuvia, jotka eivät ole normalisoituja tai segmentoituja.

Segmentointiin suhtaudutaan pessimistisesti neuroverkkoja käsittelevässä kirjallisuudessa. Segmentoinnin ongelmaa voisi yrittää kiertää Krizhevskyn ja muiden [2012] tavoin tunnistamalla useita alueita syötekentästä ja tulkitsemalla näitä tuloksia. Erilaisiin syötekentän päällä liikkuviin ikkunoihin perustuvia ratkaisuja ei olla kokeiltu kovinkaan yleisesti. Ne ovat laskennallisesti raskaita, mutta toisaalta valmiin neuroverkon tekemä tunnistus syötteestä ei modernilla laitteistolla kestä kovinkaan kauaa.

Neuroverkkojen rakenteellisia yksityiskohtia ja oppimisen tehostamista käsittelevä kirjallisuus ei ole päätynyt vielä lopullisiin totuuksin parhaista toteutustavoista. Neuroneiden sammuttaminen ja saranafunktion käyttö ovat esimerkkejä tuoreesta oppimi-

seen liittyvästä kehityksestä. Oppimista ja ylisopeutumisen välttämistä tullee myös helpottamaan tarjolla olevan käsin luokitellun datan määrän kasvu. Näytönohjainten tehokkuuden kehitys yhdistettynä suurempiin harjoitustietokantoihin voisi mahdollistaa tarkempien eli syvempien ja leveämpien neuroverkkojen kehittämisen myös vaikeisiin tehtäviin.

Viitteet

Akarachai Atakulreka and Daricha Sutivong. Avoiding local minima in feedforward neural networks by simultaneous learning. 2007. In: M. A. Orgun and J. Thornton (eds.), *AI 2007: Advances in Artificial Intelligence (Lecture Notes in Computer Science 4830),* 100-109.

Sven Behnke. 2003. *Hierarchical Neural Networks for Image Interpretation (Lecture Notes in Computer Science 2766)*. Springer.

Alberto Broggi, Pietro Cerri, Paolo Medici, Pier Paolo Porta, and Guido Ghisio. 2007. Real time road signs recognition. In: *Proc. of the 2007 IEEE Intelligent Vehicles Symposium,* 981-986.

D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber. 2011. Convolutional neural network commitees for handwritten character classification. In: *Proc. of 2011 International Conference on Document Analysis and Recognition,* 1135-1139.

D. Cireşan, U. Meier, and J. Schmidhuber. 2012. Multi-column deep neural networks for image classification. In: *Proc. of IEEE Conference on Computer Vision and Pattern Recognition 2012,* 3642-3649.

Dan C. Cireşan, Alessandro Giusti, Luca M. Gambardella, and Jürgen Schmidhuber. 2013. Mitosis detection in breast cancer histology images with deep neural networks. In: *Medical Image Computing and Computer-Assisted Intervention 2013 (Lecture Notes in Computer Science 8150),* 411-418.

Georg Edward Dahl. 2015. *Deep Learning Approaches to Problems in Speech Recognition, Computational Chemistry, and Natural Language Text Processing.* Ph. D. Dissertation, Graduate Department of Computer Science, University of Toronto.

Luiz G. Hafemann, Luiz S. Oliveria, and Paulo Cavalin. 2014. Forest species recognition using deep convolutional neural networks. In: *Proc. of 2014 22nd International Conference on Pattern Recognition.* 1103-1107.

G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580.

Geoffrey Hinton. 2012. The error surface for a linear neuron. Coursera. https://class.coursera.org/neuralnets-2012-001/lecture. Checked: 12.12.2015.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep Sparse Rectifier Neural Networks. In: *Proc. of the 14th International Conference on Artificial Intelligence and Statistics,* 315-323.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems 25 (NIPS 2012)*.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural Computation,* 1, 541-551.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. 1998a. In: *Proc. of the IEEE,* 86, 11, 2278-2324.

Yann LeCun, Leon Bottou, Genevie B. Orr, and Klaus-Robert Müller. 1998b. Efficient BackProp. In: G. Orr and K. Müller *(eds.), Neural Networks: Tricks of the Trade (Lecture Notes in Computer Science 7700),* 9-48. Springer.

Warren. S. McCulloch and Walter Pitts. 1946. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biology,* 5, 4, 115-133.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. 1986. Learning internal representations by error propagation. In: David E. Rumelhart and James L. McClelland (eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1: Foundations,* 318-362. MIT Press.

Gualtiero Piccinini. 2004. The first computational theory of mind and brain: a close look at McCulloch and Pitt's "Logical calculus of ideas immanent in nervous activity". *Synthese*, 141, 2, 175-215.

Patrice Y. Simard, Dave Steinkraus, and John C. Platt. 2003. Best practices for convolutional neural networks applied to visual document analysis. In: *Proceedings of the Seventh International Conference on Document Analysis and Recognition,* 958-963.

Matthew D. Zeiler and Rob Fergus. 2013. Visualizing and understanding convolutional networks. In: *Computer Vision – ECCV 2014 (Lecture Notes in Computer Science* 8689), 818-833. Springer.

Julkisiin tiloihin sijoitettujen suurikokoisten, vuorovaikutteisten näyttöjen tarkastelua käytettävyyden näkökulmasta

Henri Juvonen

Tiivistelmä.
Tämä tutkielma käsittelee suurikokoisten, vuorovaikutteisten näyttöjen käytettävyyttä. Tutkielmassa käsitellään julkiseen tilaan sijoitettujen järjestelmien eroa perinteisiin tietokoneisiin verrattuna sekä näiden ominaisia eroja käytettävyystutkimuksen kannalta. Tässä esitellään yleisimmät julkiset tilat, joihin järjestelmät sijoittuvat, sekä yleisimmät vuorovaikutustavat. Järjestelmien parissa tehty käytettävyystutkimus on tuottanut lupaavia tuloksia, mutta myös osoittanut puutteita sekä tutkimuksen toteutustavoissa että järjestelmien suunnittelussa.

Avainsanat ja -sanonnat: julkiset tilat, käytettävyys, näyttölaitteet, vuorovaikutteisuus.

1. Johdanto

Tietokoneen näyttö on perinteisesti esittänyt tietokoneen virtuaalista sisältöä. Tietojenkäsittelytieteen piirissä tunnetaan muitakin tapoja käyttää näyttöjä ja manipuloida niiden esittämää sisältöä. Aihetta on tutkittu jo useamman vuosikymmenen ajan ihmisen ja tietokoneen vuorovaikutus -nimen alla; jatkossa tästä käytetään englanninkielisestä nimestä *Human-Computer Interaction* johdettua lyhennystä HCI. Vuorovaikutuksen erilaisten muotojen, *modaliteettien*, lukumäärä on mittava ja tässä tutkielmassa käsitellään näistä yleisimpiä sekä joitakin uusia tai kehitteillä olevia menetelmiä. Lähtökohtana pidetään Arditon ja muiden [2015] tuoretta ja laajaa kirjallisuuskatsausta, joka esitellään vaiheittain seuraavien lukujen aikana alkaen luvusta 2. Niin ikään luvussa 2 käydään läpi alan käsitteitä, joiden avulla lukijan on helpompi ymmärtää lukemansa. Luvussa 3 luokitellaan menetelmiä, joilla näytöistä on tehty vuorovaikutteisia sekä vuorovaikutteisten näyttöjen ominaispiirteitä. Luvussa 4 syvennytään kolmeen vuorovaikutteisten näyttöjen parissa tehtyyn tutkimukseen. Luvussa 5 yhdistellään tutkielman käsittelemiä asioita sekä pohditaan ideoita tulevaisuuden tutkimukselle. Luku 6 sisältää yhteenvedon tutkielmassa käsitellyistä asioista.

2. Taustatekijöitä

Tässä luvussa esitellään käsitteitä ja tarpeellisia näkökulmia, joita lukija tarvitsee tutkielmaa lukiessaan. Ensimmäisenä tarkastellaan graafisen käyttöliittymän yleisiä piirteitä sekä sitä, miten nykyaikaisten tietokoneiden käyttöliittymä eroaa

vuorovaikutteisten näyttöjen tarjoamasta käyttöliittymästä. Tämän jälkeen tarkastellaan käytettävyyttä ja sen arviointia. Valtaosin käytettävyyden teoriapohja on pätevä myös vuorovaikutteisten näyttöjen yhteydessä, mutta eri julkaisuissa on tehty erityishuomioita liittyen nimenomaisesti julkisissa tiloissa sijaitsevaksi suunniteltaviin näyttöihin. Tästä johtuen vuorovaikutteisten, suurikokoisten näyttöjen yhdistämisen esiin nostamat erityishaasteet ovat esitettyinä samassa kohdassa.

2.1. Tietokoneista vuorovaikutteisiin näyttöihin

Graafisen käyttöliittymän tapaa esittää sisältöä kutsutaan *WIMP*-käyttöliittymäksi [van Dam 2000]. *WIMP* on lyhenne, joka tulee sanoista *windows*, *icons*, *menus* sekä *pointer devices* (ikkunat, ikonit, valikot sekä osoitinlaitteet eli hiiri ja näppäimistö), mikä ilmentää tietokoneen käyttötapaa melko tarkasti. Nämä neljä elementtiä ovat graafiselle käyttöliittymälle ominaisia tapoja esittää tietokoneen sisältö käyttäjälle. Kansiot ja tiedostot kuvataan erilaisina ikoneina ja kansioiden sisältö sekä ohjelmat taas esitetään ikkunassa. Ohjelmakohtaisesti ikkunoilla on valikko, jossa käyttäjälle tarjotaan sovellukselle ominaisia toimintoja. Ikkuna voidaan mieltää eräänlaisena virtuaalinäyttönä fyysisen näytön sisällä, mikä sisältää jonkin ohjelman tai sen osan.

On esitetty perusteluja sille, miksi WIMP olisi epäsopiva lähestymistapa vuorovaikutteisten näyttöjen yhteyteen. Yksi syy tähän on joidenkin toteutusten sallima osoittimen poissaolo, kuten on esimerkiksi kosketusnäyttöjen yhteydessä. Osoittaminen tapahtuu suoraan kohdetta koskettamalla, jolloin tietysti kohdetta myös ”osoitetaan”. Lisäksi tähän liittyy visuaalinen haaste: käyttäjän osoittaessa näytön kohteita ne peittyvät osoittimen, esimerkiksi etusormen alle. Näin ollen osoittimen esitys on ylimääräistä.

Van Dam [2000] ilmaisi jo varhain näkemyksensä WIMP-käsitteen vanhanaikaisuudesta. Tärkein syy tähän on WIMPin pohjautuminen hiiri-näppäimistöyhdistelmän avulla tapahtuvaan vuorovaikutukseen, joka on van Damin [2000] mukaan ympäristö, jossa hyödynnetään ainoastaan näkökykyä sekä alkukantaista kosketusta. Kokeneemmat käyttäjät pitävät hiirellä osoittamista turhauttavan hitaana ja turvautuvat suoraviivaisempiin näppäinoikopolkuihin. Kyseisiä näppäinyhdistelmiä taas koskee sama pulma kuin ennen graafisia käyttöliittymiä vallinneita tekstikäyttöliittymiäkin. Komennot tulee muistaa ulkoa tai palauttaa mieleen jostakin tallenteesta, mikä mitätöi osan toiminnan tehokkuudesta. Vaikka ihmiset kokevat keskuudessaan luonnolliseksi viestiä puhuen, elehtien, kuunnellen ja katsellen, on tietokoneiden yhteydessä ollut lähes mahdotonta hyödyntää tätä useista samanaikaisista aistimuksista koostuvaa tiedon-

välityskanavaa. Van Dam [2000] esittikin tulevaisuuden käyttöliittymien ja vuorovaikutuksen nojaavan entistä enemmän luonnollisen vuorovaikutuksen menetelmiin. Arditon ja muiden [2015] katsaus osoittaa näin käyneen ja he kertovat *post-WIMP* -metaforasta sekä sen mukanaan tuomista uudenlaisista vuorovaikutusmuodoista perinteiseen tietokoneen käyttöön verrattuna. Vuorovaikutustapoja eli modaliteetteja käsitellään tarkemmin luvussa 3.

2.2. Käytettävyydestä

HCI tutkii ihmisen ja tietokoneen vuorovaikutusta, siis laitteiden käyttöä ja sen ohessa tapahtuvaa laitteen ja käyttäjän välistä viestintää yleisesti. Eräs tutkimusaloista keskittyy mittaamaan käytettävyyttä, jonka kansainvälinen standardointiorganisaatio ISO (*International Organization for Standardization*) määrittelee seuraavasti: käytettävyys ilmentää, miten jotakin (esimerkiksi tuotetta tai laitetta) voidaan käyttää määrättyjen käyttäjien toimesta siten, että he saavuttavat tavoittelemansa lopputuloksen ottaen huomioon toiminnan vaikuttavuuden, tehokkuuden ja käyttäjän tyytyväisyyden käytön yhteydessä. [ISO 9241-11 1998; Bevan 2006] Toisaalta tutkimuksen kohteena on laitteen käyttö ja sen vaikutukset niin ihmisen mieleen kuin toimintaankin. Lisäksi tutkimuksen kohteena ovat myös toiminnan vaikutus laitteeseen sekä mahdolliset uudet toimintatavat, jotka syntyvät tämän teknologian käyttöönoton myötä ja jälkeen. Tätä uusien toimintatapojen syntymistä voisi pitää tietojenkäsittelytieteelle luonteenomaisena, sillä usein tietoteknologisten sovellusten tai apuvälineiden käytön myötä niille syntyy täysin uusia käyttötapoja ja niiden yhdistelmiä, kuten on tapahtunut internetin sekä älypuhelinten myötä. Kännykän käytöstä on tullut monipuolisempaa, kun sen toiminnot ovat laajentuneet uuden tiedonsiirtomuodon myötä.

2.3. Käyttökokemus

Norman [2013] käsittelee järjestelmän käytöstä syntyvää kokemusta. Hänen mukaansa tärkein seikka jo järjestelmän suunnittelussa on toimintojen *näkyvyys*, joka koostuu viidestä käsitteestä. *Affordanssit* ovat käyttäjän havaittavissa olevia käyttömahdollisuuksia – käyttäjän tulkintoja järjestelmän käyttötarkoituksesta. Usein affordanssi tulisi *merkitä*. Merkintä voi olla suunnittelijan luoma, kuten kehotus koskea näyttöä käytön aloittamiseksi. Merkintä voi olla kuitenkin myös tahaton, kuten aiempien näyttöä koskettaneiden sormenjäljet.

Merkintä johdattaa käsitteisiin *kartoitus* ja *palaute*, jotka liittyvät läheisesti toisiinsa. Kartoitus on oikeanlaisen syötteen edellytys, ilman tätä käyttäjän on arvattava toimintojen seuraus. Syötettä seuraava palaute luo kartoituksen edellytykset. Palautteen on seurattava syötettä viiveettä ja oltava informatiivinen. Pa-

lautetta ei tulisi myöskään antaa taukoamatta. Vaarana on, että käyttäjä häiriintyy tai hän ei koe palautetta merkitykselliseksi ja sivuuttaa myös tarpeelliset viestit. Normanin [2013] mukaan huonosti suunniteltu palaute voi olla jopa haitallisempaa kuin palautteen puuttuminen kokonaan.

Kun affordanssi on näkyvästi merkitty ja toiminnasta saa palautetta, on käyttäjälle järjestelmän käytöstä syntyvä *käsitteellinen malli* yksinkertaisempi ennakoida suunnittelussa. Käsitteellinen malli on selitys käyttäjän toimintatavalle ja ilman mallia toiminta on joko ulkoa opettelua tai kokeilemista. Käsitteellinen malli voi olla hyvin yksinkertainenkin, ja kun käyttäjän käsitteellinen malli on samankaltainen suunnittelijan mallin kanssa, ei käyttäjän tarvitse perehtyä järjestelmään syvällisesti sitä käyttääkseen. Tällöin käyttäjälle syntyvä käyttökokemus on positiivinen, jolloin voidaan myös sanoa järjestelmän käytettävyyden olevan hyvä. Käyttökokemus on kuitenkin subjektiivinen ilmiö, mistä syystä mittavälineiden kehitys on monisyistä.

2.4. Käytettävyyden mittaaminen

Erääksi vuorovaikutteisten näyttöjen käytettävyystutkimuksen erityishaasteeksi julkisen sijainnin myötä on muodostunut käytön tarkkailu, kun perinteisesti käytettävyystutkimus on tapahtunut kontrolloidussa ympäristössä. Chen ja muut [2011] rakensivat kosketusnäyttötutkimustansa varten erityisen tilan, jossa käyttäjien toimia tallentaa kuusi eri kameraa. Samoin kameroita käyttivät Zhang ja muut [2015] GazeHorizonia tutkiessaan. Videotallenteiden analysointi on työlästä tutkimuksen ollessa pitkäkestoinen [Heimonen et al. 2014]. Erona on yksittäisen testihetken suunniteltu ajankohta sekä kesto, joita ei julkisissa paikoissa ole mahdollista ennalta arvioida. On myös muita haasteita, muun muassa Ylipulli ja kumppanit [2013] sekä Chen ja muut [2011] mainitsevat pelon itsensä nolaamisesta yleisön edessä. Erityisesti uusien vuorovaikutustapojen yhteydessä tämä on merkittävää, sillä jotkin kokevat olevansa ikään kuin näyttämöllä, jos heidän ympärille kerääntyy yleisöä. Zhang ja muut [2015] taas raportoivat GazeHorizonin yhteydessä honey pot -ilmiöstä. Ilmiöllä tarkoitetaan tilannetta, jossa ihmiset kerääntyvät katsomaan, mitä muut katsovat. Tämä luo käyttäjälle puitteet pelätä julkinolaamista [Ylipulli et al. 2013], mutta kaikki eivät koe tällaista ilmiötä samoin. Edelleen Zhang ja kumppanit [2015] kertovat ilmiön myös synnyttäneen sosiaalisia vuorovaikutustilanteita, joissa tarkkailtu käyttäjä alkaa selostaa ympärillään oleville järjestelmän käyttöä. Tämä on omiaan uuden vuorovaikutustavan omaksumiselle.

Käytettävyyden mittaamiseksi on kehitetty erilaisia menetelmiä – eräs tärkeimmistä on System Usability Scale -niminen kyselylomake eli *SUS*. Järjestel-

män kehitti vuonna 1986 John Brooke. Kysely perustuu 10 numeroituun väitteeseen, joista parittomat ovat negatiivissävytteisiä ja parilliset myötäileviä. Nämä kymmenen väitettä valikoituivat alkuperäisten 50 väitteen joukosta sen mukaan, mitkä ovat parhaiten erottelevia kahden tutkimuksessa käytetyn järjestelmän välillä. Hän kuitenkin lisää huomautuksen, että väitteet yhdessä antavat yksittäisen numeroarvon, joka kuvastaa tarkasteltavana olevan järjestelmän käytettävyyttä. Yksinään mikään väitteistä ei ole merkittävä. [Lewis and Sauro 2009]

Eräs toinen käytettävyyden mittaustapa on käyttökokemusta kartoittava User Experience Questionnaire -lomake eli UEQ. Tämä perustuu kuuden seikan mittaamiseen: kokeeko käyttäjä järjestelmän vetovoimaiseksi, kuinka ymmärrettävä järjestelmä on käyttäjän mielestä, kokeeko käyttäjä voivansa toimia järjestelmän parissa tehokkaasti, järjestelmän luotettavuus, stimulaatio ja uutuus. [Laugwitz et al. 2008] Sekä SUS että UEQ ovat luvun 4 käsittelemissä järjestelmissä käytettyjä kyselymalleja.

3. Vuorovaikutteiset näytöt

Vuorovaikutteisuus on tutkielman keskiössä. Kuten johdannossa mainitaan, erilaisia näyttöjä on valjastettu muun muassa tiedonvälitystoimeen sekä huomionkeräämiseen, muun muassa mainontaan. Vastaavasti modaliteetteja on kehitelty pidemmälle myös kuluttajamarkkinoilla.

Ardito ja muut [2015] ovat luokitelleet julkaisuja sen mukaan, minne tutkimuksen kohteena ollut laite on sijoitettu. Niistä 43 % sijaitsee laboratorioissa eli ei-julkisessa käytössä. Jäljelle jääneistä Ardito ja muut ovat erottaneet toisistaan seuraavat sijainnit: kaupungit sekä toimistot, joihin sijoittuu kolmannes heidän katsauksensa kohteista, oppilaitokset peruskoulusta korkeakouluihin ja yliopistoihin, erilaiset tapahtumapaikat, niin kutsuttu ”kolmas paikka” kuten kahvilat, kulttuurikohteet sekä kaupat.

Julkisiin paikkoihin sijoitellut vuorovaikutteiset näytöt ovat aiemmin olleet harvinaisia muun muassa laitekustannusten vuoksi. Uudet vuorovaikutusmuodot ovat teknisesti erilaisia perinteisiin osoitinlaitteisiin, hiireen ja näppäimistöön verrattuna siten, että tietokoneen on oltava jatkuvassa valmiudessa ja reagoitava viiveettä. Aihetta käsitellään tarkemmin kohdassa 3.4.

3.1. 1960- ja 1970-luvut: esihistoriaa

Historia on nykyhetken mahdollistaja, mistä syystä tässä luvussa käydään lyhyesti läpi tärkeimmiltä osin vuorovaikutteisten näyttöjen kehityspolku kuvaputkimonitoreista rakennuksen julkisivun kokoisiin seinäinstallaatioihin. Kohtien 3.1., 3.2. ja 3.3. tarkoitus on saattaa lukijalle tutuksi nämä kehitysvaiheet. Lukijaa,

joka ei ole historiallisesta näkökulmasta erityisen kiinnostunut, ohjataankin jatkamaan kohdasta 3.4.

HCI-tiedeyhteisössä ollaan oltu kiinnostuneita digitaalisella näytöllä esitettävien kohteiden osoittamisesta manipuloinnista ilman erillisiä syötelaitteita jo usean vuosikymmenen ajan. Tämä käy ilmi mainitusta Arditon ja muiden [2015] julkaisusta, joka tarkastelee 206 julkaisua aiheesta suuret, vuorovaikutteiset näytöt. Aikaisimmat kokeilut kosketusnäytöistä ovat 1960-luvulta [Johnson 1965], jossa lennonjohtotehtäviä suunniteltiin voitavan hoitaa osittain näiden avulla. Toteutus pohjautuu kuvaputkimonitoriin, jonka lasipinnalla sijaitsevat kuparijohdot ilmaisevat monitoria ohjaavalle tietokoneelle, jos jokin koskettaa näyttöä ja lisäksi kosketuskohdan näytöllä. Mielenkiintoisena yksityiskohtana tämän yhteydessä esitetyn teknologian raportoidaan olevan edelleen käytössä. Nykyisiä toteutuksia lähentelevä plasmanäyttö PLATO IV hyödynsi infrapunalähettimiä ja -antureita kosketustapahtuman rekisteröimiseksi [Sherwood 1972].

3.2. 1980-luku: kosketusvuorovaikutuksen kehitys ja standardointi

1980-luvulla HCI:n parissa työskentelevät tutkimus- ja työryhmät ovat alkaneet kehitellä tarkempia tekniikoita kosketuksen havaitsemiseksi ja havainnon täsmentämiseksi, kuten monen näyttöä yhtäaikaisesti koskevan sormen havainnointi (*multitouch*, jatkossa monikosketus [Lee et al. 1985]) sekä kosketusten paine-erojen mittaaminen [Minsky 1984]. Lisäksi käytettäviä käsi- ja sormieleitä on ryhdytty standardoimaan sekä luomaan uusia modaliteetteja, kuten kehon liikkeet [Krueger 1985]. Samanaikaisesti on tutkittu, mitä tehtäviä tällä uudenlaisella tekniikalla voidaan suorittaa ja pohdittu toimintoja, jotka eivät ole olleet mahdollisia ennen vuorovaikutteisia näyttöjä sekä toimintoja, jotka eivät vieläkään ole optimaalisia joko vaadittujen suoritteiden tai teknologian johdosta. Esimerkiksi näytöllä esitettyjen, kolmiulotteisten objektien manipulointiin liittyy tällaisia seikkoja, kuten Minsky [1984] kertoo. Kaksiulotteisen tai korkeintaankin rajoitetun kolmiulotteisen [Grossman and Widgor 2007] esityksen käsittely, kuin se olisi kolmiulotteinen, käy haastavaksi liikuttaessa näytön normaalin suuntaisesti. Näytön normaaliksi kutsutaan suoraa, joka on 90 asteen kulmassa näytön lasipintaan nähden. Rajoitetuksi kolmiulotteisuudeksi nimitetään toteutusta, jossa esitetyt kohteet ovat joko esitetty kolmiulotteisina tai niiden välinen suhde perustuu kolmiulotteisuuteen. Esimerkkinä perinteinen ikkunointijärjestelmä: ikkunoiden asettelu tapahtuu näytön normaalin suhteen, mutta käyttäjän ei ole mahdollista siirtää käyttämäänsä ikkunaa toisen alle. Vain ei-aktiiviset ikkunat voivat sijoittua tällä tavoin.

3.3. 1990- ja 2000-luvut: prototyypeistä kaupallisiksi tuotteiksi

1990-luvun alkuun sijoittuu Weiserin [1991] artikkeli, jossa esitetään tietojenkäsittelyn tulevaisuuden näkymiä. Julkaisu on eräs merkittävimmistä suunnannäyttäjistä ja se sisältää aikaansa nähden hyvin edistyksellisiä toteutuksia. Esimerkiksi Weiser mainitsee tietojenkäsittelystä yleisesti, että ”todellinen voima kumpuaa laitteiden välisestä vuorovaikutuksesta”. Näin muodostui pohja ”kaikkialla olevalle”, ubiikille tietojenkäsittelylle (*ubique computing*) sekä ubiikkiuden toteuttamisen vaatimille, monipuolisen erikokokoisille tietokoneille. Weiser [1991] ilmaisi, että tietynlaiset ja tietynkokoiset tietokoneet soveltuvat parhaiten tietynlaisiin tehtäviin sen sijaan, että kaikenlaiset koneet sopisivat yhtäläisesti kaikenlaisiin tehtäviin.

1990-luvulla syntyivät myös pöytätasot vaihtoehtona pystyasennossa oleville seinänäytöille. Eräs esimerkki tällaisesta pöydästä on Digital Desk, jonka yhteydessä esiteltiin käsinkosketeltavaa vuorovaikutusta (*tangible interaction*). Digital Desk perustuu virtuaalisen ja fyysisen työpöydän yhtäläisyyksiä korostavaan tietokoneistetun todellisuuden lähestymistapaan. Sen luonnehditaan tähtäävän ”tietokonemaisten toimintojen lisäämistä fyysisiin kohteisiin ja ympäristöihin siten, että vuorovaikutus niiden kanssa tapahtuu kuin ennenkin”. [Wellner 1991] Näkökulma on merkittävä työnteon kulttuuria ajatellen, mutta hyödyllinen ottaa huomioon myös kehitettäessä digitaalisia vastineita fyysisille esineille. Fyysisen ja virtuaalisen ympäristön välisen kuilun kaventamiseen tähtää myös 1990-luvulla kehitetty käsinkosketeltava käyttöliittymä (*graspable user interface*), jossa vähintään yhtenä syötelaitteena hyödynnetään virtuaaliesineeseen viittauksen sisältävää ”palikkaa” (*bricks*). [Fitzmaurice et al. 1995] 2000-luvulla vuorovaikutteisia näyttöjä on ilmestynyt katukuvaan [Ylipulli et al. 2013] ja muun muassa Sharp [2012] ja Microsoft [2015] ovat esitelleet vuorovaikutteisen näytön.

3.4. Vuorovaikutuksen modaliteetit

Luvun 3 aiemmissa kohdissa on mainittu joitakin modaliteetteja. Tässä esitellään vuorovaikutteisten näyttöjen päämodaliteetit. Ardito ja kumppanit [2015] ovat jaotelleet katsauksensa julkaisut vuorovaikutustapojen mukaan ja erottavat toisistaan neljä kategoriaa: kosketusvuorovaikutus, vuorovaikutus erillisen laitteen avulla, vuorovaikutus käsinkosketeltavan esineen avulla sekä kehollinen vuorovaikutus, joka käsittää kaikenlaiset eleet, katseen käytön sekä esimerkiksi käyttäjän läsnäolon havainnoimisen.

Vuorovaikutteisten näyttöjen tukemia modaliteetteja luokitellessa tulee huomata, että toistaiseksi pääosan järjestelmistä raportoidaan tukevan vain yhtä mo-

daliteettia kerrallaan. Erilaisia hybridimahdollisuuksia on lukuisia ja eräs esimerkki tällaisesta hybridimodaliteetista on luvussa 4 esiintyvä kosketusvuorovaikutusta tukeva CubIT-järjestelmä, jossa järjestelmään kirjautumisessa hyödynnetään kosketeltavaa esinettä sekä henkilökohtaisen mediakirjaston luomiseen erillistä laitetta.

3.4.1. Kosketus

Kosketusvuorovaikutus on ensimmäisiä suoran vuorovaikutuksen modaliteetteja, joita tietokoneille on kehitetty. Aiemmin mainittu Johnsonin [1965] artikkeli lennonjohdon tehtäviin suunnitellusta kosketusnäytöstä on ensimmäisiä suunnannäyttäjiä. Koskettaminen ja sormella osoittaminen ovat ihmiselle luontaisia tapoja fyysisten esineiden manipulointiin ja niiden parissa navigointiin.

Kosketusnäytön teknisiä toteutustapoja on lukuisia. Tyypillisin laite, jossa tavallinen kuluttaja kohtaa kosketusnäytön, on tabletti tai älypuhelin. Näissä kosketus tunnistetaan sormen ja laitteen välillä kulkevasta sähköimpulssista. Suuremmissa laitteissa on perinteisesti käytetty infrapunalaitteita tai videokameraa kosketuksen tulkintaan. Arditon ja muiden [2015] katsauksessa kerrotaan projektorin ja projektion heijastavan pinnan käytön yleistyneen vuoden 2004 jälkeen. Tutkijat arvelevat syynä olevan edullisen toteutuksen, mutta huomauttavat samassa yhteydessä edullisuuden riippuvan viime kädessä toteutuksesta. Projektio voidaan esittää joko heijastavan pinnan takaa tai edestä heijastaen. Tämä on eräs kustannuksia muokkaavista tekijöistä: edestä heijastaen voidaan käyttää valkoista tai muuta vaaleaa seinää heijastepintana, mutta takaa heijastettuna projektorin vaatima tila estää tämän, ja silloin tarvitaan erillinen heijastepinta sekä sen taakse tilaa projektor(e)ille. Koska jälkimmäinen tapa on kuitenkin mahdollista toteuttaa yhtenä suljettuna kokonaisuutena, saattaa sen kokonaiskustannukset vaihdella toimittajasta ja sijoituskohteesta riippuen paljon. Takaprojektiota käytettäessä käyttäjän oma varjo ei peitä näytöllä esitettyjä asioita. Myöhemmin esitellään muutama erityishuomio liittyen näytön edestä tapahtuvaan projektioon ja sisällön esitykseen.

3.4.2. Kehollinen vuorovaikutus

Liikkeentunnistus ja -havainnointi ovat tavallisia muun muassa pelikonsoleissa. Nintendon markkinoille tuomat liikeanturein varustetut ohjaimet saivat kaksi muuta konsolivalmistajaa kehittämään liikkeentunnistukseen perustuvaa ohjausta. Sony seurasi Nintendoa ja julkisti oman versionsa liikkeen tunnistavista peliohjaimista ja lisäsi myös perinteisiin ohjaimiin liikeanturit. Microsoft kehitti kameratekniikkaan perustuvan Kinect-laitteen, joka havaitsee sen edessä olevat

ihmiskehot ja niiden liikkeet toteuttaen näin ohjaimettoman pelaamisen. Tällaisesta Kinectin tukemasta vuorovaikutuksesta on kyse, kun käsitellään kehollista vuorovaikutusta, kun taas peliohjaimen käyttö vastaa alakohdan 3.4.4. esittelemää tapaa vuorovaikuttaa toisen laitteen avulla. Kinectiä käytetään Information Wall -järjestelmässä [Mäkelä et al. 2014] Tampereen yliopistossa. Järjestelmä sijaitsee yhdessä yliopiston sisääntuloauloista ja se esittää kontekstiin soveltuvaa sisältöä, esimerkiksi viereisen kahvion ruokalistan sekä paikallisten tapahtumien ilmoituksia. Käyttäjä osoittaa näytön kohteita kädellään, joka ilmentyy näytöllä kursorina. Osoittamisessa hyödynnetään magneettisen kursorin tekniikkaa, jolloin kursori hakeutuu kohti lähintä objektia. Näin käyttäjän kehon liikkeitä seuraavan laitteen, tässä Kinect, tuottamia mittavirheitä saadaan minimoitua. Osoitus tulkitaan edelleen valinnaksi, jos käyttäjä osoittaa jotakin pidemmän aikaa.

Kehollinen vuorovaikutus on Kinectiä laajempi käsite. Siihen sisältyy kehon ulokkeiden käytön lisäksi kasvojen tunnistaminen sekä kasvojen avulla vuorovaikuttaminen, kuten Tuiskun ja muiden [2013] Face Interface -prototyypissä. Face Interface on vaihtoehtoista vuorovaikutusta tutkiva koeasetelma, joten se käsitellään vuorovaikutuksen osalta. Käyttäjä osoittaa katseellaan näytöllä olevan virtuaalinäppäimistön kirjaimia ja suorittaa valinnan nostamalla poskipäitä tarkoituksenaan kirjoittaa sana “aurinko”. Käyttäjä pitää päässänsä silmälasien kaltaista kehystä, joka kantaa kahta kameraa: toinen kuvaa käyttäjän silmää ja seuraa sen liikkeitä. Toinen kamera seuraa käyttäjän näkemää näkymää. Lisäksi laitteessa on infrapuna-anturi, joka kuvaa sarveiskalvon heijastumia, jota käytetään katseen suunnan laskelmassa. Jotta poskipäillä tehtävä valinta onnistuu, on kehyksissä elektroniset anturit kasvonliikkeiden havaitsemiseksi. Tällaiset järjestelmät ovat kömpelöitä, sillä ne vaativat käyttäjäkohtaisesti kalibroidun seurantalaitteen.

Luvussa 4 esitellään Zhangin ja muiden [2015] toteuttama GazeHorizon-järjestelmä, joka hyödyntää katseentunnistamista tavallisen web-kameran turvin. GazeHorizon tukee osoitus ja valinta -menetelmää tehden eron sisällön selaamisen ja sisällön staattisen esittämisen välillä sen mukaan, mitä käyttäjä tekee. Suuri ero on Face Interfacen edellyttämä käyttäjäkohtainen laitteiston kalibrointi. GazeHorizon suunniteltiin niin, että sen nähdessään kuka vain voi käyttää laitetta.

3.4.3. Kosketeltavan esineen avulla tapahtuva vuorovaikutus

Käsinkosketeltava käyttöliittymä tarjoaa monipuoliset mahdollisuudet, jotka käsittävät sekä näytön esittämien objektien manipuloinnin jonkin erillisen esineen avulla että myös käyttäjälle esitetyn näkymän manipuloinnin tai käyttäjän tun-

nistamisen. Ensimmäisestä yksinkertaisin esimerkki on styluksen käyttö erityisesti sellaisissa tilanteissa, joissa näyttö ei reagoi sormin koskettamiseen. Jo mainittu DigitalDesk [Wellner 1991] on toinen esimerkki kosketeltavasta vuorovaikutuksesta, johon on ammennettu ominaisuuksia työpöytä-metaforasta. DigitalDesk koostuu pöydästä, johon on liitetty kamera sekä projektori. Kamera kuvaa pöydällä olevat fyysiset kohteet ja esimerkiksi skannaa pöydällä olevat dokumentit. Projektori esittää järjestelmän palautteita sekä virtuaalisia kohteita pöydällä, kuten laskimen. Pöydällä olevia kohteita, olivatpa ne sitten virtuaalisia projektioita tai fyysisiä esineitä, käyttäjä voi taas koskettaa sormellansa, jolloin kamera välittää kyseisen kosketustapahtuman tietokoneelle. Koska yläpuolella sijaitseva kamera ei helposti kykene erottamaan paikallaan olevaa sormea pöytää napauttavasta sormesta, pöydässä on mikrofoni, jonka signaalista järjestelmä varmentaa klikkaamisen. Nykyaikaista kohteen siirtoa paikasta toiseen sormella vetäen pöytä ei tue, sillä mikrofoni ei kykene erottamaan tästä syntyvää ääntä.

Eräs esimerkki kosketeltavan esineen avulla tapahtuvasta vuorovaikutuksesta on näkymän muokkaaminen erillistä esinettä käyttäen. Kim ja Elmqvist [2012] esittelevät järjestelmän, jonka esittämä sisältö on muokattavissa erityisillä tunnisteilla varustetuilla "linsseillä". Kim ja Elmqvist käyttävät tutkimuksessaan linssinä ohutta, jonkin verran valoa lävitseen päästävää paperia sekä piirtoheitinkalvoa. Linssiä on tarkoitus pitää projektorin ja heijastavan pinnan välissä ja siitä heijastuva osa näytön esittämistä asioista sisältää vaihtoehtoista sisältöä tavanomaiseen nähden – tässä tietoja myytävistä asunnoista linssin rajaamalla alueella. Järjestelmä havaitsee linssin tunnisteen infrapunakameran avulla ja yksilöi ne, jolloin käyttäjä voi asettaa eri linssille eri ominaisuudet. Linssejä voi myös vapaasti yhdistellä keskenään.

3.4.4. Toisen laitteen avulla

Toisen laitteen avulla tapahtuva vuorovaikutus käsittää järjestelmät, joita ohjataan etänä esimerkiksi mobiililaitteella. Lisäksi kategoriaan kuuluvat järjestelmät, joissa erillistä laitetta käytetään esimerkiksi tekstinsyöttöön, vaikka järjestelmä muuten toimisi muun modaliteetin mukaisesti. Luetellessaan vuorovaikutteisille näytöille ominaisia toimintoja Cardoso ja José [2014] kertovat esimerkkejä erilaisista syötelaitteista ja -tavoista, joita on käytetty vuorovaikutteisten näyttöjen parissa. Perinteisten syötelaitteiden toimintaa hyvin lähellä on kameralla varustetun kännykän käyttö hiiren korvikkeena. Kameran tehtävä on välittää ohjelmalle jatkuvaa kuvavirtaa, josta ohjelma laskee suunnan ja nopeuden osoittimen liikkeelle kuvassa tapahtuvien muutosten myötä. Mobiililaitteiden nimeämistä on myös käytetty syötteenantotekniikkana. Bluetoothin edellyttämä

laitteiden pariutus tarjoaa mahdollisuuden antaa syötteitä oman laitteen nimitunnisteessa, jolloin nimi ja sen muutokset välittyvät pariutuksen yhteydessä ja sen jälkeen järjestelmälle. Koska syötelaite on näissä tapauksissa toinen tietokoneenkaltainen järjestelmä vuorovaikutteisen näytön lisäksi, ovat laajennusmahdollisuudet loputtomat.

WallSHOP on järjestelmä, joka koostuu vuorovaikutteisesta näytöstä sekä web-sovelluksesta [Masuko et al. 2015]. Mobiililaitteen avulla käyttäjät voivat ladata (*pull*) laitteeseensa näytöllä esitettyjä asioita, kaupan tuotteita, ladata (*push*) laitteestansa näytölle personoimansa tuotteen tai arvostelun jostakin tuotteesta. Lisäksi eräs tekijöiden ajatuksista oli, että näin toimien ostosten tekemisestä tulisi yhteisöllisempää, sillä yhtä näyttöä käyttävät useat henkilöt näkevät toisensa sekä fyysisesti että näytöllä kursorin ilmaisemana.

4. Vuorovaikutteisia näyttöjä julkisissa tiloissa

Seuraavaksi tarkastellaan kolmea esimerkkiä vuorovaikutteisista näytöistä ja niiden käyttötutkimuksista. Esimerkit valikoituivat siten, että näitä ei valittu Arditon ja muiden [2015] katsaukseen. Syitä tähän ovat esimerkiksi myöhemmin tapahtunut julkaisu tai järjestelmän jokin erityislaatuinen ominaisuus. Näiden parissa suoritettu tutkimus tarjoaa tärkeitä vihjeitä uudenlaisten järjestelmien suunnitteluun.

4.1. CubIT

Koko seinän täyttävä CubIT-järjestelmä on Australiassa, Brisbanessa sijaitsevan Queensland University of Technologyn Cube-nimisessä opetus- ja näyttelytilassa. Järjestelmä tukee monikosketusvuorovaikutusta sekä käyttäjän tunnistautumista RFID-sirulla varustetulla kortilla. Lisäksi järjestelmään liittyy mobiilisovellus ja verkkoalusta, joiden avulla käyttäjä voi lisätä sisältöä järjestelmän tarjoamaan käyttäjäkohtaiseen säiliöön. Eräs järjestelmän suunnitelluista käyttötarkoituksista on käyttäjän luoman sisällön esitys ja käsittely. Järjestelmän suunnitellut ominaisuudet ovat monen käyttäjän yhtäaikainen tuki ja helppo sisällön lataus sekä jakaminen käyttäjäkohtaisesta säiliöstä toiseen tai näytölle yleisön tarkasteltavaksi. [Rittenbruch 2015]

Järjestelmän tapa esittää sisältöä näytöllä noudattaa metaforaa *messy desktop*, jossa näytön kohteet ovat kaikkien järjestelmää kyseisellä hetkellä käyttävien käytettävissä. Kohteet ovat vapaasti liikuteltavissa, kierrettävissä ja suurennettavissa aina maksimikokoonsa asti. Nämä mahdollistavat sellaisia yhteistyöskentelyn muotoja, joita ei aiemmissa toteutuksissa ole ollut. Järjestelmän oltua käytössä yhdeksän kuukauden ajan tutkijat selvittivät järjestelmän aikaansaamaa

käyttökokemusta kyselyllä, johon oli yhdistetty myös SUS- ja UEQ-kyselylomakkeet. [Rittenbruch 2015]

Lähes kaikki vastaajat (n=48) olivat tunnistaneet järjestelmän perustoiminnot, joina työryhmä pitää käyttäjäksi rekisteröitymistä ja sisäänkirjautumista sekä mediasisällön latausta ja esittämistä. Yli puolet vastaajista kertovat suorittaneensa sisällönhallinnallisia toimintoja seinän avulla, joita ovat sisällön poistaminen, esittäminen sekä kiinnittäminen johonkin kohtaan näyttöä suhteessa käyttäjän omaan työtilaan. Järjestelmän päätoimintoihin lukeutuva sisällön jakaminen vetäen ja pudottaen suoraan työtilasta toiseen oli toiminto, jota moni ei ollut suorittanut. [Rittenbruch 2015]

Käyttökontekstia hahmottelevan osan mukaan yleisin käyttötapa oli käyttäjän tallentaman sisällön esittäminen, joko seinällä yleisesti, kollegoille, ulkopuolisille tilassa vieraileville tai osana esitystä. Vastaajista viidesosa kertoi käyttäneensä järjestelmää osana konferenssiesitystään ja kaksi kertoi pitäneensä luennon järjestelmän avulla. [Rittenbruch 2015]

4.2. GazeHorizon

Zhang ja kumppanit [2015] suunnittelivat ja toteuttivat Lancasterin yliopistossa näytön, jonka tarkoituksena on toimia vuorovaikutuksen aloittajana. Tutkijat varustivat GazeHorizonin virtuaalisin opastein ohjeistaakseen käyttäjää uudenlaisen järjestelmän käyttöön. Ohjeet johdattavat ohikulkijan sijaintiin, josta järjestelmän käyttö onnistuu. Käyttäjän läsnäolon havaittuaan järjestelmän ohjeet häipyvät ja näytölle ilmaantuu sisältö, jota selata. Käyttäjän käännyttyä pois järjestelmä palaa alkutilaan ja on valmis seuraavaa käyttäjää varten.

Kuten aiemmin on mainittu, tavanomaisesti katseenseuranta edellyttää käyttäjäkohtaista kalibrointia seurantalaitteistolle, jotta vuorovaikutus onnistuu. GazeHorizonissa käytetään tavanomaista web-kameraa katseen seuraamiseksi. Järjestelmä tunnistaa kameran kuvasta käyttäjän pupillit ja mittaa niiden suhteellista etäisyyttä silmäkulmasta vaakatasossa, silmäkohtaisesti. Tästä etäisyydestä kone laskee pupilli-silmäkulma-suhteella (*Pupil-Canthi Ratio, PCR*) kulman, jossa käyttäjän silmät ovat suhteessa näytön äärirajoihin. Kulman myötä kone laskee kohdan, johon käyttäjä katsoo. Menetelmä toimii pupillien vaakatasossa tapahtuvien liikkeiden tulkitsemiseen, mutta pystyakselilla tapahtuvan liikkeen tunnistaminen on haasteellista muun muassa silmäluomien rakenteesta johtuen. Käyttäjän katsoessa alas silmäluomet peittävät pupillit, jolloin järjestelmä tulkitsee käyttäjän lopettaneen käytön. Sen sijaan sivusuunnassa liikkuessaan pupillit pysyvät kameran näkymässä, jolloin järjestelmä toimii. Käyttäjän katsoessa keskelle hänen tulkitaan tarkkailevan mielenkiintonsa kohdetta ja näyttö esittää passiivisesti sisältöä. Näytöllä esitetään elokuvaesitteitä ja käyttäjä

voi katsomalla kohti näytön reunaa etsiä sellaisiakin esitteitä, joita ei näytöllä juuri sillä hetkellä ole esitettynä. Elokuvavalikoima on esitetty kehärakenteisena listana, jota voi selata loputtomasti uudelleen. [Zhang et al. 2015]

Käyttötutkimus oli kolmivaiheinen. Ensimmäisessä osassa tarkasteltiin käytön vaatimuksia, joiden pohjalta järjestelmään tehtiin parannuksia. Yksi tarkastelun kohteista oli opastuksen tarve: ohikulkijoita kutsuttiin kokeilemaan järjestelmää ja heille kerrottiin vaiheittain järjestelmän käytöstä, jos käyttäjä ei keksinyt, miten laitetta käyttäisi. Ensimmäisen vaiheen myötä tutkijat päättivät lisätä ohjeistuksen näytölle esitettäväksi. Toisessa vaiheessa testattiin järjestelmään lisättyjen ohjeiden toimivuutta. Tämän vaiheen tulokset osoittivat, että järjestelmään tulisi lisätä myös ohjeistus käyttäjän sijaintiin, sillä käyttäjät jäivät usein seisomaan liian kauas, eikä kamera havainnut silmiä oikein.

Kolmannessa vaiheessa järjestelmä asetettiin yliopiston aulaan ilman opastajaa, jotta tutkijat ymmärtäisivät, kuinka ihmiset toimivat järjestelmän opasteiden avulla. Vaihe toteutettiin kahdessa osassa. Ensin järjestelmä oli käytettävissä kaksi päivää, jonka aikana tehtyjen haastattelujen perusteella järjestelmää vielä paranneltiin. Jälkimmäinen osuus kesti niin ikään kaksi päivää, mutta tällöin GazeHorizoniin ei enää tehty muutoksia. Osallistujia ei kutsuttu käyttämään järjestelmää eikä käyttötuokioita häiritty. Järjestelmää käyttäneitä haastateltiin jälkikäteen ja järjestelmän keräämiä lokitietoja käyttötuokioista käytettiin tulosten tulkinnassa hyväksi. Valtaosa haastatelluista (76,0 % ensimmäisessä osassa, 85,4 % jälkimmäisessä osassa) onnistui käyttämään järjestelmää sen tarjoamien ohjeiden avulla. Jotkut onnistuivat myös siitä huolimatta, että he sivuuttivat opasteet. [Zhang et al. 2015]

4.3. Oulun UBI-näytöt

Oulun yliopisto ja Oulun kaupunki toteuttivat yhteishankkeen, jossa Oulun keskustaan asennettiin vuorovaikutteisia näyttöjä sisä- ja ulkotiloihin yhteensä 18 kappaletta. Tutkimuksen yhtenä tarkoituksena oli arvioida käytettävyyden lisäksi, omaksuvatko kaupunkitilan käyttäjät näyttöjen tarjoamat palvelut osaksi arkeaan. Näytöt tukevat kosketusvuorovaikutusta. Lisäksi käyttäjä voi yhdistää puhelimensa näyttöön Bluetoothin, NFC-sirun, QR-koodin tai tekstiviestin avulla, jolloin käyttäjä voi palvelun salliessa siirtää puhelimestaan sisältöä näytölle tai toisinpäin. Kaikki palvelut ovat käytettävissä ilman erityisiä laitteita tai kirjautumista. Palveluita ovat muun muassa uutisointi, viihde ja pelit sekä maakuntakohtaiset tapahtumatiedotteet. Osa palveluista on väliaikaisia, kuten kävelykadun kunnostamiseen liittyvä viestintä. [Ylipulli et al. 2013]

Järjestelmän käyttöloki osoittaa käytön vähentyneen hiljalleen laitteiden asennuksen jälkeen. Käyttökokemuksen kartoittamiseksi laadittiin kysely, joka

kohdistettiin kahdelle eri opiskelijaryhmälle: lukiolaisille sekä yliopiston opiskelijoille. Vastausten tärkeimmät huomiot koskevat näyttöjen käyttöastetta sekä koettua hyötyä. Kyselyyn vastanneista lukiolaisista 48 % (n=98) oli käyttänyt näyttöjä, yliopiston opiskelijoista 46 % (n=924). Vastausten suuri ero on vastaajaryhmän kokema hyöty: lukiolaisista 78 % piti järjestelmää ja sen palveluita hyödyllisinä, kun taas yliopisto-opiskelijoista vain 28 % oli tätä mieltä. [Ylipulli et al. 2013]

Ymmärtääkseen syvemmin järjestelmän synnyttämää käyttökokemusta tutkijat järjestivät kaksi lisätutkimusta kahden erityyppisen ihmisryhmän parissa: nuoret aikuiset ja vanhukset. Nuorten aikuisten parissa suoritettu tutkimus sisälsi neljä ryhmäkeskustelua, joiden perusteella laadittiin käyttäjän itse täytettävä päiväkirja. Päiväkirja sisälsi kymmenen erilaista tehtävää. Koska vähemmistö heistä oli käyttänyt järjestelmää aiemmin, oli sen kokeilu yksi suoritettavista tehtävistä. Osallistujien (n=48) oli tarkoitus päiväkirjan avulla pohtia omaa tieto- ja viestintätekniikan käyttöä. Vanhuksille sen sijaan järjestettiin haastattelutilaisuuksia, joissa keskusteltiin heidän näkemyksistään tieto- ja viestintätekniikan käytöstä, kännykän yleistymisestä sekä kaupunkitilan käytöstä. Koska vain neljä (n=16) oli käyttänyt järjestelmää, ei sen käyttöä käsitelty keskusteluissa kovinkaan paljon. [Ylipulli et al. 2013]

5. Pohdinta

Ardito ja muut [2015] kertovat vuorovaikutteisten näyttöjen suunnittelun olevan haasteellista ennalta-arvaamattoman laajan käyttäjäkunnan vuoksi muun muassa siksi, että käyttäjät ovat eri ikäisiä ja heidän taitonsa sekä aiemmat kokemuksensa ovat erilaisia. Luvun 4 esimerkit vahvistavat tätä käsitystä. Haastattelujen sekä käyttölokin perusteella UBI-näyttöjen käyttö on ollut vähäistä. Ylipulli ja muut [2013] arvioivat syitä olleen useita. Heidän mukaansa järjestelmän käyttö on vähäistä muun muassa sen uutuuden vuoksi. Uuden teknologian omaksuminen tapahtuu suhteessa jo omaksuttuihin, ja tavanomaisesti pienikokoiset näytöt ovat vuorovaikutteisia ja suuret eivät. Tällöin suurikokoinen, vuorovaikutteinen näyttö sotkee käyttäjän käsitteellistä mallia. Jaotteluun liittyy myös käytön yksityisyysaste. Vanhempien haastateltujen henkilöiden vastauksissa on erityisen selkeä viesti yksityisten asioiden käsittelystä julkisella paikalla. Useat heistä välttävät puhelimenkin käyttöä julkisesti. Perusteluksi esitettiin, ettei ole soveliasta muita kohtaan altistaa heitä toisten henkilökohtaisille asioille. Yleisemmin vastaajat kuitenkin kertoivat toimintaa tarkkailemaan jäävien ihmisten häiritsevän käyttöä. Kyseessä on honey-pot-ilmiön varjopuoli, pelko itsensä nolaamisesta julkisella paikalla. Toisaalta Zhangin ja muiden [2015] tulkinnan mukaan ilmiö voi synnyttää myös positiivista sosiaalista vuorovaikutusta. He mainitsevat

käyttäjien ryhtyneen opettamaan sivustakatsojia laitteen käyttöä selittäen ääneen, mitä tekevät. Tämä nopeuttaa käytön omaksumista ja viestii järjestelmän affordansseista.

Affordanssien merkinnän tärkeyden osoittaa GazeHorizonin käyttöä koskevat tutkimustulokset. Myös aloittelijat onnistuivat käyttämään järjestelmää, mikä kertoo ohjeistuksen onnistuneen. Zhang ja muut [2015] kertovat yksinkertaisenkin opasteen, kuten look here -kehotteen olevan riittävän tehokas. Kehote auttaa käyttäjää eliminoimaan muut kuin katsevuorovaikutuksen järjestelmää koskevasta käsitemallistansa. Tärkeää on myös reagointi katseeseen, tässä ohjeen poistuminen. Tämä välitön palaute osoittaa käyttäjälle hänen toimineen oikein. Osa vastaajista kertoi luovuttaneensa yhden käyttöyrityksen jälkeen. Merkinnän sijoittelulla on myös merkitystä. Aluksi lattiassa ollut merkintä oikeasta käyttöetäisyydestä sivuutettiin pääosin kokonaan. Käyttäjät kertoivat asemoineensa itsensä näytön esittämän videokuvan avulla sen sijaan, että olisivat noudattaneet tekstiohjeistusta asettua metrin päähän näytöstä.

CubITin käyttöä koskevassa tutkimuksessa todettiin käyttäjien käyttäneen vain joitakin järjestelmän toimintoja. Esimerkiksi yhdeksi päätoiminnoksi suunniteltua median jakamista käyttäjältä toiselle ei juurikaan oltu käytetty. Syitä Rittenbruch [2015] kertoo olevan muun muassa ohjeistuksen puute. Järjestelmän formaalia käyttöopastusta ei ollut tarjolla kuin verkosta luettavan käyttöohjeen myötä. On käyttäjän oman mielenkiinnon varassa koettaa uusia toimintoja tällaisessa tilanteessa. Lisäksi uudenlainen vuorovaikutus on tässäkin tapauksessa käyttäjää haastava.

On perusteltua vedota Normanin [2013] esittelemiin käyttöliittymäsuunnittelun käsitteisiin myös vuorovaikutteisten näyttöjen suunnittelussa, kuten yllä oleva osoittaa. Affordanssin merkitseminen edesauttaa käyttäjää luomaan uudesta järjestelmästä oikeanlaisen käsitteellisen mallin ja välttämään epäjohdonmukaisia tilanteita. Erityisesti uudenlaiset toiminnot sekä modaliteetit edellyttävät merkintää – muussa tapauksessa näiden käyttö on lähes mahdotonta. Ylipullin ja muiden [2013] suorittamat haastattelut eri ikäryhmiä edustavien käyttäjien parissa osoittavat lisäksi, kuinka eri ikäiset ihmiset kokevat uudenlaiset, julkiset järjestelmät eri tavoin tarpeellisiksi. Tavanomaisesti uusien järjestelmien kohderyhmän ajatellaan koostuvan nuorista aikuisista. Tällöin on vaarana, että esimerkiksi ikäihmiset ja heidän tarpeensa jäävät huomioimatta. Julkinen tila, esimerkiksi kaupunki, on eri ihmisryhmien myötä monipuolisessa käytössä ja tämä tulisi ottaa huomioon suunnittelutyössä.

6. Yhteenveto

Vuorovaikutteiset näytöt julkisissa tiloissa tarjoavat uudenlaisia käyttömahdollisuuksia uudessa ympäristössä. Järjestelmät tarjoavat myös uudenlaisia tapoja vuorovaikuttaa virtuaalisisällön kanssa. Näistä uusista ominaisuuksista johtuen järjestelmän suunnitteluun tuleekin kiinnittää erityistä huomiota ja pohtia, kuinka käyttäjä opastetaan näitä käyttämään. Ilman opastusta on mahdollista, ettei jokin ominaisuus tule koskaan käytetyksi. Lisäksi opastuksen puute vaikeuttaa järjestelmän käyttöä, vaikka toiminnot olisivat tunnettuja muussa yhteydessä.

Erityisesti suunnittelussa tulee kiinnittää huomiota järjestelmän kohderyhmään. Julkisissa tiloissa tämä ryhmä on erittäin kirjava. Koska eri ihmisryhmien tarpeet ja käyttökokemukset ovat erilaisia, ei ole yhtä ainoaa tapaa jolla järjestelmän omaksuminen osaksi päivittäistä toimintaa tapahtuu. Tämä korostaa käytettävyyden arvioinnin tarpeellisuutta.

Käytettävyyden arvioimiseksi on kehitetty julkisiin paikkoihin soveltuvampia menetelmiä, jotka eivät vaadi valvottua koeympäristöä, kuten perinteinen käytettävyystutkimus. Näistä saatujen tulosten avulla järjestelmän synnyttämää käyttökokemusta voidaan kohentaa entisestään. Perinteisiä menetelmiä voidaan kuitenkin osittain soveltaa, kun otetaan huomioon uusien ominaisuuksien mukanaan tuomat uudet vaatimukset.

Viiteluettelo

Carmelo Ardito, Paolo Buono, Maria Francesca Costabile and Giuseppe Desolda. 2015. Interaction with large displays: a survey. *ACM Comput. Surv*. 47, 3, Article 46, 38 pages.

Nigel Bevan. 2006. International standards for HCI. In: Claude Ghaoui (ed.), *Encyclopedia of Human Computer Interaction.* Idea Group Publishing, 362-372.

Jorge C. S. Cardoso and Rui José. 2014. Interaction tasks and controls for public display applications. In: *Advances in Human-Computer Interaction*, Volume 2014, Article ID 371867, 17 pages.

Chien-Hsu Chen, Hsiao-Mei Hung, I-Jui Lee, Yu-Wen Chen and Fong-Gong Wu. 2011. Observe the user interactive behavior with a large multi-touch display in public space. In: Constantine Stephanidis (ed.), *Proc. of the 6th International Conference on Universal Access in Human-Computer Interaction: Context Diversity - Volume Part III*. Springer. 141-144.

Andy van Dam. 2000. Beyond Wimp. *Vision 2000*. IEEE Computer Graphics and Applications. 50-51.

Edgar. A. Johnson. 1965. Touch display — a novel input/output device for computers. *Electronics Letters* 1, 8, 219–220.

KyungTae Kim and Niklas Elmqvist. 2012. Embodied lenses for collaborative visual queries on tabletop displays. *Information Visualization* 11, 4, 319-338.

Myron W. Krueger, Thomas Gionfriddo and Katrin Hinrichsen. 1985. VIDEOPLACE: an artificial reality. In: *Proc. of the SIGCHI Conference on Human Factors in Computing Systems (CHI'85).* ACM, 35–40.

Bettina Laugwitz, Theo Held and Mertin Schrepp. 2008. Construction and evaluation of a User Experience Questionnaire. In: A. Holzinger (ed.), *USAB 2008,* LNCS 5298, Springer, 63-76.

SK Lee, William Buxton and Kenneth C. Smith. 1985. A multi-touch three dimensional touch-sensitive tablet. In: *Proc. of the SIGCHI Conference on Human Factors in Computing Systems (CHI'85).* ACM, 21-25.

James R. Lewis and Jeff Sauro. 2009. The factor structure of the System Usability Scale. In: Masaaki Kurosu (ed.), *Human Centered Design 5619*, Springer, 94-103.

Soh Masuko, Masafumi Muta, Keiji Shinzato and Adiyan Mujibiya. 2015. Interactive study of WallSHOP: multiuser connectivity between public digital advertising and private devices for personalized shopping. In: *Proc. of the 4th International Symposium on Pervasive Displays*, ACM, 187-193.

Microsoft. 2015. Surface Hub. Press release. http://news.microsoft.com/2015/06/10/microsoft-surface-hub-available-for-businesses-to-order-on-july-1/. Checked 18.12.2015.

Margaret R. Minsky. 1984. Manipulating simulated objects with real-world gestures using a force and position sensitive screen. *Computer Graphics* 18, 3, 195-203.

Ville Mäkelä, Tomi Heimonen, Matti Luhtala and Markku Turunen. 2014. Information wall: evaluation of gesture-controlled public display. In: *Proc. of the 13th International Conference on Mobile and Ubiquitous Multimedia*, ACM, 228-231.

Donald A. Norman. 2013. *The Design of Everyday Things : Revised and Expanded Edition*. Basic Books.

Markus Rittenbruch. 2015. Supporting collaboration on very large-scale interactive wall surfaces. *Computer Supported Cooperative Work (CSCW)*, 24, Springer, 121–147.

Sharp. 2012. AQUOS Board. Press release. http://www.sharpusa.com/AboutSharp/NewsAndEvents/PressReleases/2012/January/01_09_CES_2012_80-Inch_AQUOS_Board.aspx. Checked 18.12.2015.

Bruce Sherwood. 1972. Status of PLATO IV. *SIGCUE Outlook 6*, 3, 3-6.

Outi Tuisku, Veikko Surakka, Ville Rantanen, Toni Vanhala and Jukka Lekkala. 2013. Text entry by gazing and smiling. *Advances in Human Computer Interaction,* 2013, Hindawi Publishing Corporation, 13 pages.

Mark Weiser. 1991. The Computer for the 21st century. *Scientific American*, 265, 94-104.

Pierre Wellner. 1991. The DigitalDesk calculator: tangible manipulation on a desk top display. In: *Proc of the 4th ACM Symposium UIST '91*, ACM, 27-33.

Johanna Ylipulli, Tiina Suopajärvi, Timo Ojala, Vassilis Kostakos and Hannu Kukka. 2013. Municipal WiFi and interactive displays: appropriation of new technologies in public urban spaces. *Technological Forecasting & Social Change* 89, 145–160.

Yanxia Zhang, Ming Ki Chong, Jörg Müller, Andreas Bulling and Hans Gellersen. 2015. Eye tracking for public displays in the wild. *Personal Ubiquitous Computing*, 19, Springer, 967-981.

Yksinkertaiset hierarkkiset klusterointimenetelmät

Valtteri Kostiainen

Tiivistelmä

Hierarkkinen klusterointi on ohjaamattoman oppimisen menetelmä, jota käytetään luokitteluongelmien ratkaisemiseen. Tässä tutkielmassa tarkastellaan kolmea hierarkkisen klusteroinnin menetelmää: lähin naapuri, kauimmainen naapuri ja keskiarvo. Tutkimus tehtiin menetelmiä käsittelevän kirjallisuuden avulla. Menetelmiä tarkastellaan yksinkertaisten esimerkkien avulla.

Tutkielman luvuissa kaksi, kolme ja neljä esitetään ja avataan koneoppimisen, tiedonlouhinnan ja hierarkkisen klusteroinnin käsitteet. Hierarkkisen klusteroinnin käsite avataan yleisesti menemättä yksittäisten menetelmien tasolle. Luvussa viisi perehdytään tarkemmin lähin ja kauimmainen naapuri menetelmiin ja keskiarvomenetelmään.

Avainsanat: Ohjaamaton oppiminen, klusterointi, lähin naapuri -menetelmä, kauimmainen naapuri -menetelmä, keskiarvomenetelmä.

1 Johdanto

Koneoppiminen ja varsinkin erilaiset luokittelumenetelmät ovat suurten tietomäärien aikakaudella välttämättömiä työkaluja. Koneoppimisella tarkoitetaan järjestelmän kykyä oppia uutta tietoa tai toimintatapoja kokemuksen perusteella, ilman lisäohjeita tai ohjelmointia. Koneoppiminen tapahtuu siis järjestelmän itsenäisen työskentelyn kautta ja sillä hallitaan ongelmia, joita ihminen ei voi käsin ratkaista. Yhtenä esimerkkinä voidaan ajatella verkkokauppojen suositusjärjestelmiä, jotka käyttävät suosituksissaan käyttäjän

tekemiä aiempia valintoja ja samankaltaisten käyttäjien valintoja. Olisi tehotonta räätälöidä käsin miljoonia algoritmeja eri käyttäjien suosituksia varten. Koneoppimisen avulla järjestelmät itse kehittävät säännöt suosituksia varten. [10]

Mitchelin [7] mukaan koneoppiminen tieteenalana pyrkii vastaamaan seuraavaan kysymykseen: kuinka voimme rakentaa tietokonejärjestelmiä, jotka automaattisesti edistyvät kokemuksen myötä ja mitä ovat pohjimmaiset lait jotka määrittelevät kaikki oppimisprosessit? Mitchell määrittelee koneoppimisen tarkemmin seuraavalla tavalla: Kone oppii ja ottaa oppiessaan huomioon tehtävän T, suoritustekijän P ja kokemuksen tyypin E. Jos järjestelmä parantaa suoritustaan P tehtävässä T luotettavasti, seuraa kokemus E. Riippuen tekijöiden T, P ja E määritelmistä, oppimistehtävää voisi myös kutsua tiedonlouhinnaksi, itsenäiseksi tutkinnaksi, tietokantapäivitykseksi tai esimerkkiohjelmoinniksi.

Tämän tutkielman tarkoitus on esitellä lyhyesti kolme yksinkertaista hierarkkista klusterointi -menetelmää. Tutkielman tarkoituksena on antaa lukijalle tietoa hierarkkisesta klusterointiprosessista ja sen sijoittumisesta laajempaan koneoppimisen ja tiedonlouhinnan kontekstiin.

Tutkielma on kirjallisuuskatsaus, joka rakentuu alan peruskirjallisuuden varaan. Luvuissa 2 ja 3 avaan koneoppimisen ja tiedolouhinnan käsitteet. Neljännessä luvussa esittelen hierarkkisen klusteroinnin periaatteet yleisellä tasolla. Luvussa 5 esittelen tarkemmin lähin naapuri ja kauimmainen naapuri -menetelmät ja keskiarvomenetelmän. Lisäksi suoritan yksinkertaisen klusteroinnin edellä mainituilla menetelmillä.

2 Koneoppiminen

Koneoppimisen metodit jaetaan karkeasti kolmeen kategoriaan, ohjaamattomaan, ohjattuun ja vahvistusoppimiseen. Ohjatussa oppimisessa käsiteltävän datan luokat tiedetään ennakkoon, jolloin järjestelmälle voidaan antaa palautetta sen tekemistä valinnoista. Järjestelmän tehtävänä voi olla esimer-

kiksi erottaa henkilöautot, pakettiautot, kuorma-autot ja bussit toisistaan. Aina kun kone tekee päätöksen, prosessin valvoja voi arvioida sen ja antaa järjestelmälle siitä palautetta. Tätä oppimisprosessissa käytettävää aineistoa kutsutaan opetusjoukoksi (training set). Opetusjoukon muodostaminen on yksi ohjatun oppimisen heikkouksista, sillä opetusjoukon koko, jotta se olisi tarpeeksi tehokas, voi olla kymmeniätuhansia tietoalkioita. Järjestelmä kehittää päätöksentekoprosessiaan itsenäisesti valvojalta saamansa palautteen perusteella, eli oppii. Käytännössä järjestelmä muodostaa datan avulla säännöt, joiden perusteella se tekee valintansa. Esimerkkejä ohjatun oppimisen menetelmistä ovat päätöspuut (decision trees) ja Bayesin verkko (Bayesian belief network). [10] Ohjattua oppimista sovelletaan luokitteluun ja regressioanalyysiin. Käytyään opetusjoukon läpi järjestelmä kykenee luokittelussa antamaan diskreetin arvon analysoimilleen uusille tapauksille. Regressioanalyysissa järjestelmä antaa arvon tutkittavalle tapaukselle opetusjoukon perusteella. Esimerkkinä regressioanalyysista on tilanne, jossa järjestelmä antaa kiinteistöille arvoja aiempien kiinteistökauppojen tietojen perusteella. [10]

Ohjaamattomassa oppimisessa oppimisprosessin lopputulosta ei tiedetä, eikä järjestelmä saa palautetta tekemistään valinnoista. Ohjaamattomassa oppimisessa datasta johdettavia luokkia ei tunneta ennen kuin järjestelmä on käsitellyt analysoitavan datan. [10] Analysoitava data voi esimerkiksi käsitellä jonkin yrityksen asiakastietoja. Asiakastiedoista voidaan halutaa selvittää, muodostavatko asiakkaat asiakasryhmiä. Tiedot syötetään järjestelmään, joka soveltaa tilanteeseen sopivaa klusterointialgoritmia. Tuloksena järjestelmä antaa asiakasryhmät, sekä määräävät tiedot kustakin ryhmästä. Järjestelmä voi myös olla löytämättä tulosta. Ohjaamattomassa oppimisessa ei siis tarvita suuria opetusjoukkoja, koska oppiminen tapahtuu puhtaasti dataa analysoimalla. [10] Ohjaamattoman oppimisen huono puoli on se, että lopputuloksen ennustettavuus on heikkoa, eli opittu asia voi olla väärä [2].

Vahvistusoppimisen luonteeseen pääsee parhaiten käsiksi tarkastelemalla sitä ohjaamattoman oppimisen kautta. Ohjaamattomassa oppimisessa järjestel-

mälle kerrotaan opetusjoukon käsittelyn aikana, mihin luokkaan tutkittavat tietoalkiot kuuluvat. Vahvistusoppimisessa järjestelmän saama palaute ei ole yhtä tarkkaa kuin ohjaamattomassa oppimisessa. Tarkkojen luokkatietojen sijasta järjestelmälle annetaan palautetta *vahvistussignaalin* (reinforcement signal) avulla. Vahvistussignaali on positiivinen, kun järjestelmä tekee oikeita valintoja, mikä ohjaa valintoja oikeaan suuntaan. [2]

3 Tiedonlouhinta

Tiedonlouhinta ja koneoppiminen ovat käsitteinä sisäkkäisiä ja hankalia erottaa toisistaan. Tiedonlouhinta voidaan ajatella koneoppimisen sovellusalana. Siinä tiedonlouhija käyttää koneoppimisjärjestelmiä, jokin tavoite mielessään, jollekin aineistolle. Krzysztof ja muut [2] tiivistävät tiedonlouhinnan käsitteen seuraavalla tavalla:

> ”Tiedonlouhinnan tarkoituksena on saada jotakin järkeä suuriin määriin, enimmäkseen valvomatonta dataa jollakin sovellusalalla”

Valvomattomalla datalla tarkoitetaan tässä yhteydessä dataa, joka muodostuu syötteistä, joille ei ole tunnettua tarkoitusta tai vastausta. Tiedonlouhintaprosessi tehdään yhteistyössä sovellusalan ammattilaisten kanssa, mikä mahdollistaa tiedonlouhintajärjestelmän sovittamisen erikseen kulloiseenkin sovellusalaan. [2]

Tiedonlouhintamenetelmiä sovelletaan suuriin datamääriin, sillä pienempiä datamääriä voidaan käsitellä muilla tekniikoilla tai manuaalisesti. [2] Käsiteltävät tiedot voivat olla lähes minkä tyyppisiä tahansa, kuten dataa osakemarkkinoiden toiminnasta tai multimediaa. [11] Tiedonlouhinnassa käytettävää tietoa säilytetään tietokannoissa, joita yhdistelemällä muodostuu tietovarastoja. Tiedonlouhintamenetelmiä voidaan luokitella usealla tavalla. Zaïane [11] esittää seuraavia kriteerejä luokittelun perusteiksi:

- louhittava tiedonlähde

- käytetty tietomalli
- löydetty tieto
- käytetty tiedonlouhintamenetelmä.

Krzysztof ja muut [2] ja Zaïane [11] asettavat tiedonlouhinnan tuloksille kriteerejä. Jos kriteerit jäävät täyttymättä, tiedonlouhinnan tulos on heidän mukaansa triviaalia. Krystof ja muut pitävät tärkeimpänä tiedon ymmärrettävyyttä. Parhaimmillaan ymmärrettävyys tarkoittaa sitä, että louhitusta datasta voidaan päätellä jokin selkeä malli, joka voidaan ilmaista loogisten sääntöjen kautta. Zaïane sen sijaan painottaa tiedon selkeää visualisointia ymmärtämisen apuna. Louhinnan tuloksena muodostuneen mallin tulee olla myös pätevä ja hyödyllinen. Oikeellisuuden ja hyödyllisyyden määrittelevät sovellusalan asiantuntijat. Jos malli ei sisällä uutta tietoa, se on käytännössä turha datan omistajille.

4 Hierarkkinen klusterointi

Hierarkkinen klusterointi on koneoppimisen menetelmä, jota käytetään luokitteluongelmien ratkaisemisessa [4]. Se on ohjaamatonta oppimista ja poissulkevaa, eli datasta syntyviä luokkia ei tunneta ennakkoon ja tietoalkiot voivat kuulua vain yhteen luokkaan [4]. Cai ja muut [1] määrittelevät klusteroinnin menetelmäksi, jonka tavoitteena on, että samassa klusterissa olevat tietoalkiot ovat äärimmäisen samankaltaisia ja samankaltaisuus eri klustereissa olevien tietoalkioiden välillä on minimaalinen.

Hierarkkisessa klusteroinnissa klusterit muodostuvat alkutilanteessa yksittäisistä luokittelemattomista tietoalkioista. Klusteroinnin tuloksena muodostuu hierarkkinen rakennelma, jonka tasot ovat klusteroinnin vaiheita [2]. Klusterien väliset suhteet esitetään etäisyysmatriisina [6]. Dataa voidaan säilyttää myös monikossa $x = (x_1, ...x_k)$, jossa on k tietoalkion ominaisuuksia

vastaavaa lukuarvoa [3]. Jos objektiparien välille ei ole määritelty merktseviä etäisyyksiä, klusterointi ei ole mahdollista [4].

Hierarkkiset klusterointimenetelmät voidaan jakaa karkeasti kahteen ryhmään. Kokoavissa menetelmissä tietoalkiot muodostavat ensimmäisellä tasolla kukin oman klusterinsa ja *tarkkuus* (granularity) on suurimmillaan. Viimeisellä tasolla kaikki tietoalkiot ovat samassa klusterissa ja tarkkuus on pienimmillään. [1, 2, 6] Jakavat menetelmät toimivat päinvastaisella periaatteella. Alkuvaiheessa tietoalkiot ovat samassa klusterissa ja niitä jaetaan vaiheittain tarkempiin klustereihin, kunnes kukin tietoalkio muodostaa oman klusterinsa [6]. Kokoavat menetelmät ovat huomattavasti yksinkertaisempia kuin jakavat menetelmät ja tästä syystä yleisempiä [6].

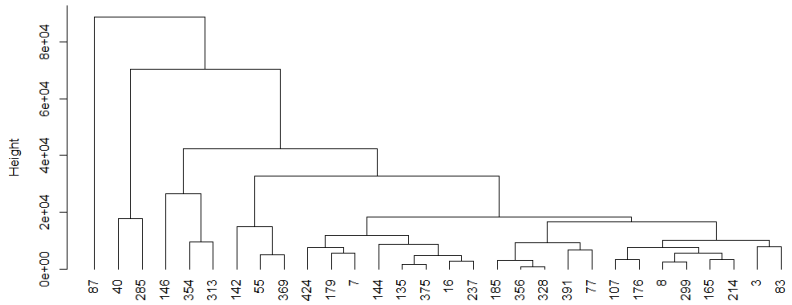
Klusterointimenetelmät voidaan jakaa myös graafimenetelmiin ja geometrisiin menetelmiin. Graafimenetelmissä, kuten lähin naapuri -menetelmässä, klusteria edustaa aligraafi. Geometrisissä menetelmissä sitä edustaa klusterin keskipiste. Aligraafin olemus ja keskipisteen laskutapa riippuvat valitusta klusterointimenetelmästä. [4] Tässä tutkielmassa keskitytään seuraaviin kokoaviin menetelmiin: lähin naapuri- ja kauimmainen naapuri -menetelmään ja keskiarvomenetelmään.

Yleisesti klusteroinnissa tehdään etukäteen päätös siitä, kuinka moneen klusteriin tietoalkiot jaetaan. Klusterien määrä riippuu valituista menetelmistä ja sovellusalasta. Hierarkkisessa klusteroinnissa täytyykin siis määritellä, mitkä klusteroinnin tasot ovat informatiivisia ja hyödyllisiä. [1, 2]

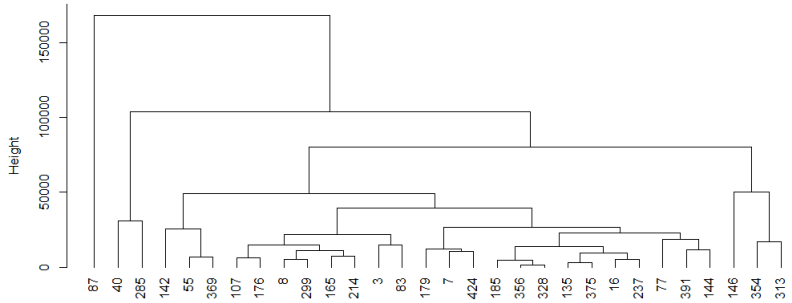
Klusterointiprosessia visualisoidaan usein dendogrammilla, sillä se on helpompi ymmärtää kuin lista abstrakteja symboleja. Dendogrammi koostuu klustereita esittävistä symboleista, joita yhdistellään viivoilla merkkinä uuden klusterin muodostumisesta. Leikkaamalla dendogrammi horisontaalisesti viivalla voidaan tunnistaa kullakin tasolla olemassa olevat klusterit. [4]

Etäisyysmatriisia laskettaessa valitaan jokin etäisyysfunktio, esimerkiksi (euklidinen etäisyys) [4]. Etäisyysfunktion valinta on oleellista klusteroinnin lopputuloksen kannalta, sillä se vaikuttaa syntyvien klustereiden muodostumiseen [2]. Valittua etäisyysfunktiota sovelletaan ensimmäisen etäisyysmatrii-

sin muodostuksessa. Myöhemmissä etäisyysmatriiseissa klustereiden väliset etäi-syydet määritetään lähin naapuri menetelmällä tai jollain muulla menetelmällä. Yleinen käytetty etäisyysluokka on Minkowskin etäisyys, josta tärkeimpiä erityistapauksia ovat euklidinen, Manhattan- ja Tšebyšovin etäisyydet. Näiden funktiot on esitetty taulukossa 4, missä i ja j ovat tietoalkioita, joiden välisiä etäisyyksiä tarkastellaan ja n piirteiden määrä klusterissa. Kuvissa 1 ja 2 näkyvissä dendogrammeissa on suoritettu hierarkkinen klusterointi samalle tietoalkiojoukolle käyttäen kauimmainen naapuri -menetelmää, mutta kuvassa 1 on käytetty euklidista etäisyyttä ja kuvassa 2 on käytetty Manhattan-etäisyyttä.



Kuva 1: Klusterointi kauimmainen naapuri -menetelmällä etäisyysfunktiona euklidinen etäisyys



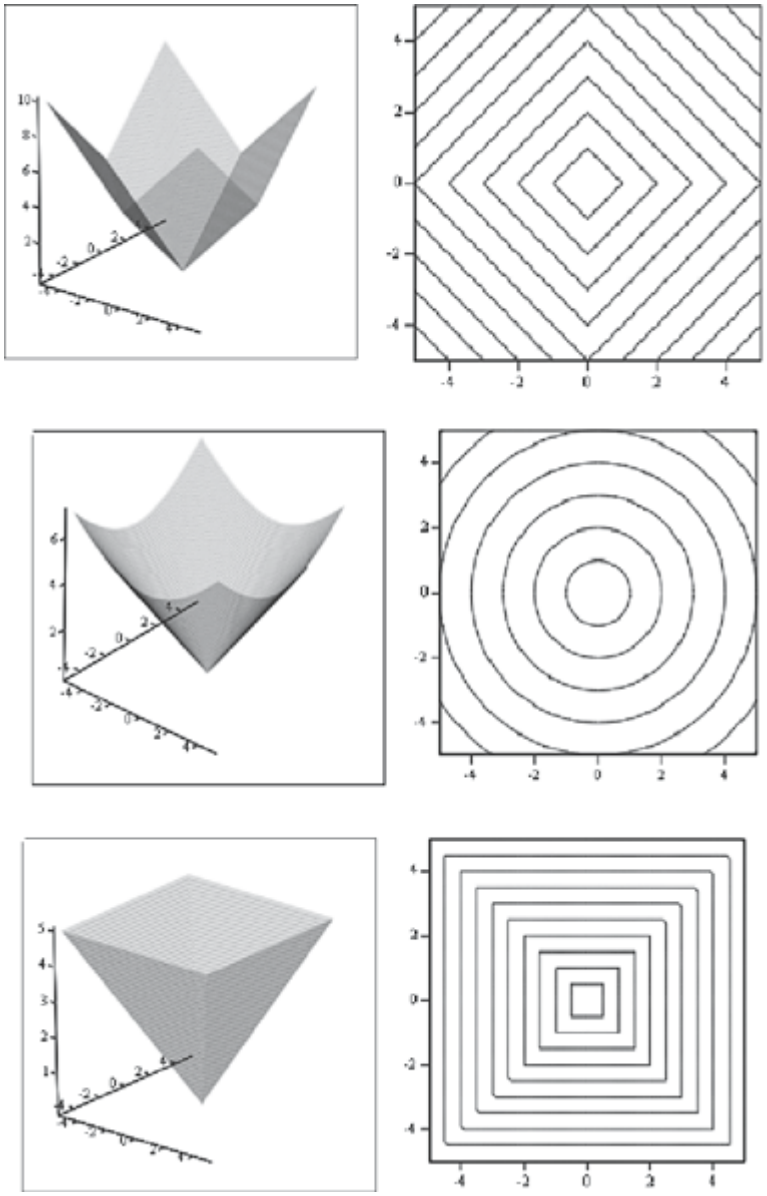
Kuva 2: Klusterointi kauimmainen naapuri -menetelmällä etäisyysfunktiona Manhattan-etäisyys

Etäisyysfunktioiden eri vaikutuksia klusterien muotoihin voidaan yleistää

Etäisyysfunktion nimi	etäisyysfunktio
Euklidinen etäisyys	$\mathrm{d(i,j)}=\sqrt{\sum_{k=1}^{n}(i_k - j_k)^2}$
Manhattan-etäisyys	$\mathrm{d(i,j)}=\sum_{k=1}^{n}\lvert i_k - j_k\rvert$
Tšebyšovin etäisyys	$\mathrm{d(i,j)}=\max_k(\lvert i_k - j_k\rvert)$

Taulukko 1: Minkowski-etäisyydet [2]

seuraavasti: euklidinen etäisyys tuottaa useimmin pallomaisia muotoja, Manhattan-etäisyys timanttimaisia muotoja ja Tšebyšovin-etäisyys sisäkkäisiä neliöitä. Kuvassa 3 havainnollistetaan näitä eroja. [2]



Kuva 3: Graafinen esitys etäisyysfunktioiden tuottamista muodoista. a) Manhattan-etäisyys, b) euklidinen etäisyys c) Tšebyšovin etäisyys. [2]

Cai ja muut [1] nimeävät yhdeksi hierarkkisen klusteroinnin hyödyistä tuloksena muodostuvan hierarkian, joka tukee ihmisen tapaa käsitellä tuntemattomien tietoalkioiden välisiä suhteita. Muina hyötyinä pidetään yksinkertaisuutta ja vähäisiä esitietovaatimuksia liittyen tutkittavan tiedon piirteisiin

[8]. Hierarkkisen klusteroinnin heikkous on menetelmien ahneus, eli se että, ne valitsevat yhdistettävät klusterit aina lokaalisti ja klusterien staattisuus. Staattisuudessa ongelma on se, että algoritmit eivät itse kykene havaitsemaan ja korjaamaan virheellisiä klusterointeja. Tätä ongelmaa on pyritty ratkaisemaan erilaisilla klustereiden jalostusmenetelmillä, jotka uudelleensijoittaisivat väärin klusteroidut alkiot. Klusterointia sotkevat myös *häiriöpisteet* (noise points), jotka ovat ominaisuuksiltaan muista pisteistä voimakkaasti poikkeavia. Näiden pisteiden vaikutus riippuu valitusta menetelmästä. Esimerkiksi lähin naapuri -menetelmä on alttiimpi tälle verrattuna kauimmainen naapuri -menetelmään. [6]

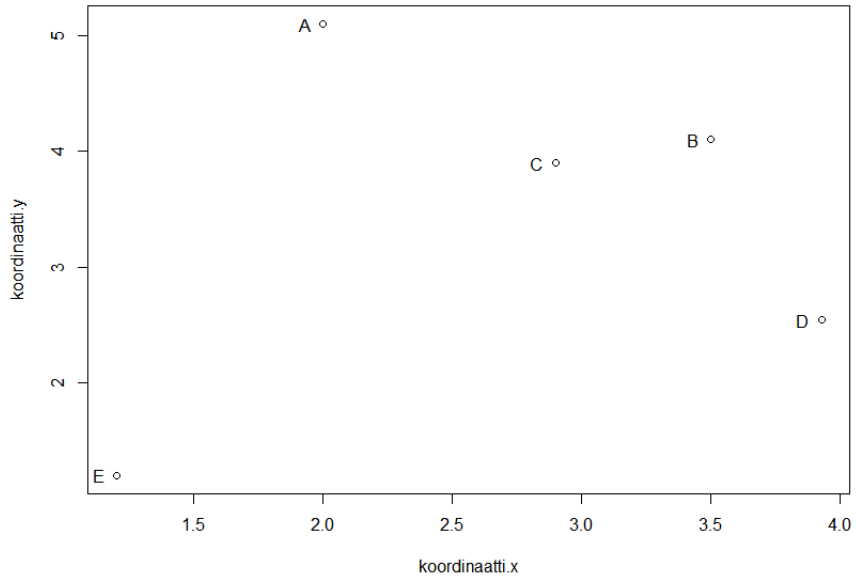
Lance ja Williams [5] esittävät yleisen algoritmin eri klusterointimenetelmille: $d_{hk} = \alpha_i d_{hi} + \alpha_j d_{hj} + \beta d_{ij} + \gamma |d_{hi} - d_{hj}|$, missä d_{xy} tarkoittaa etäisyyttä, i ja j yhdistettäviä klustereita (jos ollaan klusteroinnin alussa nämä ovat yksittäisiä tietoalkioita), k on klusteri, joka muodostuu i:n ja j:n yhdistämisestä ja h on klusteri, jonka etäisyyttä k:sta lasketaan. Parametrit α_i, α_j, β ja γ määrittävät käytetyn klusterointimenetelmän. Tätä algoritmia kutsutaan nykyään yleisesti Lancen-Williamsin algoritmiksi. Algoritmi yleistää menetelmien suurimman eron, sen kuinka eri klustereiden väliset etäisyydet lasketaan. Taulokossa 2 on esitetty kolmen eri klusterointimenetelmän parametrien arvot Lancen-Williamsin algoritmille. Taulukossa n on tietoalkioiden lukumäärä klusterissa.

Klusterointimenetelmä	α_i	α_j	β	γ
Lähin naapuri -menetelmä	$\frac{1}{2}$	$\frac{1}{2}$	0	$-\frac{1}{2}$
Kauimmainen naapuri -menetelmä	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$
Keskiarvomenetelmä	$\frac{n_i}{n_i+n_j}$	$\frac{n_j}{n_i+n_j}$	0	0

Taulukko 2: Lancen-Williamsin algoritmin parametrien arvoja eri klusterointimentelmille. [8]

5 Menetelmät

Tässä luvussa esitellään tarkemmin lähin naapuri, kauimmainen naapuri ja keskiarvomenetelmät. Kun kokoava klusterointi etenee ensimmäisestä tasosta on muodostettu ensimmäinen klusteri, jossa on enemmän kuin yksi jäsen. Klusterointimenetelmät eroavat toisistaan siinä, miten näiden useampia jäseniä sisältävien klusterien väliset etäisyydet lasketaan. Menetelmien esittelyssä tullaan soveltamaan Lancen ja Williamsin [5] käyttämiä luokittelukriteerejä ja yksinkertaista esimerkkiklusterointia. Esimerkkiklusterointi suoritetaan kuvassa 4 näkyvälle joukolle, josta muodostuu taulukossa 3 oleva etäisyysmatriisi.



Kuva 4: pisteidenjoukko

Lance ja Williams [5] luokittelevat menetelmät kolmen eri ominaisuuden perusteella: *kombinatorinen* (combinatorial) vai *ei-kombinatorinen* (non-combinatorial), *yhteensopiva* (compatible) vai *yhteensopimaton* (incompatible), *tilan säilyttävä* (space-conserving) vai *tilaa muokkaava* (space-distorting).

	A	B	C	D	E
A	0	1,80	1,50	3,21	3,98
B	1,80	0	0,63	1,62	3,70
C	1,50	0,63	0	1,71	3,19
D	3,21	1,62	1,71	0	3,04
E	3,98	3,70	3,19	3,04	0

Taulukko 3: Etäisyysmatriisi

Kombinatorisuuden määrittää se, pystytäänkö arvoja laskemaan ilman, että alkuperäistä dataa säilytetään. Oletetaan, että i ja j ovat klustereita, joiden yhdistäminen muodostaa uuden klusterin k. Jos uuden klusterin k etäisyys voidaan laskea klusterista h nykyisten tietojen perusteella, ei yhdistämisen jälkeen tarvitse säilyttää vanhoja tietoja. Ei-kombinatorisissa menetelmissä alkuperäiset tiedot tulee säilyttää myöhempiä laskutoimituksia varten. Yhteensopivissa menetelmissä elementtien ominaisuudet eivät muutu prosessin aikana, vaan niillä on yhtä paljon arvoja ja niitä koskevat samat rajoitteet. Yhteensopimattomissa menetelmissä elementtien ominaisuudet muuttuvat. Tilaa muokkaavissa menetelmissä voidaan havaita suorituksen aikana klusterien läheisyydessä tilan supistumista tai laajenemista. Kun menetelmä supistaa tilaa klusterien ympärillä, se saa klusterit ikään kuin liikkumaan muita klustereita kohti, ja päinvastoin tilaa laajentavat menetelmät saavat klusterit etääntymään toisistaan. Kun tila supistuu, todennäköisyys sille, että yksittäinen elementti liittyy vanhaan klusteriin, kasvaa. Tilan laajentuessa elementit muodostavat todennäköisemmin ympärilleen uusia klustereita.

5.1 Lähin naapuri -menetelmä

Lähin naapuri -menetelmä on vanhin tunnettu hierarkkinen klusterointimenetelmä [5]. Vaikka Lance ja Williams [5] väittivät jo 1960-luvulla metodin olevan vanhentunut, sitä on tutkittu paljon ja erilaisia siihen perustuvia klusterointialgoritmeja on runsaasti [9]. Lähin naapuri -menetelmän kaavaesitys on seuraavanlainen: $d(i,j) = min_{x \in i, y \in j} d(x,y)$, missä funktio $d(x,y)$ laskee

kahden elementin välisen etäisyyden, i ja j ovat klustereita, joita tarkastellaan, ja x ja y ovat i:n ja j:n jäseniä.

Lähin naapuri -menetelmä on kombinatorinen. Etäisyydet yhdistyvistä klustereista toisiin klustereihin on helppo laskea, koska etäisyyksien perustana ovat yksittäiset elementit. Se on myös yhteensopiva, sillä elementtien ominaisuudet eivät muutu klusteroinnin yhteydessä. Lähin naapuri -menetelmä on tilaa muokkaava, sillä klusterien saadessa uusia jäseniä ne liikkuvat muita klustereita kohti. Lähin naapuri -menetelmä onkin erityisen altis *ketjutukselle* (chaining). Ketjutuksessa menetelmä liittää agressiivisesti klustereita yhteen häiriöpisteiden kautta ja todelliset klusterit jäävät muodostumatta. [5, 6] Muodostuvaa häiriöpisteiden vääristämää klusteria voidaan kuvata *takkuiseksi* (straggly).

Lähin naapuri -menetelmä on graafimenetelmä, eli jos klusteroitava joukko esitetään graafinmuodossa, ovat klusterit tuolloin aligraafeja. Kun tietty etäisyys klustereiden välillä alittuu, liitetään ne yhteen kaarella. Lähin naapuri -menetelmän muodostama klusteri on maksiimaalisesti yhdistetty aligraafi eli yhdistetty komponentti. [4] Tämä on heikko rajoite, mikä johtaa aiemmin mainittuun ketjuuntumiseen, sillä vain yksi kaari vaaditaan kahden klusterin yhdistymiseen. [4]

Sovelletaan lähin naapuri -menetelmää kuvan 4 pistejoukolle. Pistejoukosta saadaan muodostettua taulukon 3 etäisyysmatriisi. Ensimmäisessä vaiheessa yhdistetään kaksi toisistaan lähintä klusteria, klusterit B ja C ne muodostavat klusterin BC. Uusi etäisyysmatriisi on taulukossa 4.

	A	BC	D	E
A	0	1,50	3,21	3,98
BC	1,50	0	1,62	3,19
D	3,21	1,62	0	3,04
E	3,98	3,19	3,04	0

Taulukko 4: Etäisyysmatriisi - klusterointitaso 1

Klusterin BC etäisyys klusteriin A saadaan kun valitaan BC:stä se jäsen,

jonka etäisyys klusteriin A on kaikista pienin. Tässä tapauksessa se on C, jonka etäisyys klusteriin A on 1,50. Sama toisetetaan klustereille D ja E. Seuraavaksi yhdistetään kaksi lähintä klusteria, klusterit BC ja A. Uusi etäisyysmatriisi on taulukossa 5.

	ABC	D	E
ABC	0	1,62	3,19
D	1,62	0	3,04
E	3,19	3,04	0

Taulukko 5: Etäisyysmatriisi - klusterointitaso 2

Seuraavaksi yhdistetään klusterit ABC ja D. Viimeisessä vaiheessa yhdistetään klusterit $ABCD$ ja E. Tällöin kaikki tietoalkiot ovat yhdessä klusterissa, jolloin kokoava klusteronti päättyy.

5.2 Kauimmainen naapuri -menetelmä

Kauimmainen naapuri -menetelmän ydinajatus klusterien etäisyyksistä on päinvastainen kuin lähin naapuri -mentelmässä. Kahden useammasta jäsenestä koostuvan klusterin välistä etäisyyttä määrittää niiden kauimmaisten jäsenten välinen etäisyys. Lancen ja Williamsin [5] käyttämien kriteerien mukaan kauimmainen naapuri menetelmä on samalla tavalla kombinatorinen ja yhteensopiva, kuin lähin naapuri -menetelmä. Kauimmainen naapuri -menetelmä on tilanmuokkausominaisuuksiltaan laajentava. Tämä johtuu siitä, että ehto klusterien yhdistymiselle on vahva. Kasvava klusteri ikään kuin kiskoo tietyltä säteeltä kaikki elementit jäsenikseen. Tämän säteen ulkopuolelta elementit alkavat muodostaa omia klustereitaan. Kauimmainen naapuri -menetelmän kaavaesitys on seuraavanlainen: $d(i,j) = min_{x \in i, y \in j} d(x,y)$, missä funktio $d(x,y)$ laskee kahden elementin välisen etäisyyden, i ja j ovat klustereita joita tarkastellaan ja x ja y ovat i:n ja j:n jäseniä. [5]

Kauimmainen naapuri -menetelmä pohjautuu graafeihin ja sen muodostamat klusterit ovat luonteeltaan maksimaalisen täydellisiä aligraafeja. Maksi-

maalisen täydellinen aligraafi on aligraafi, jonka jokaista solmua yhdistää kaari ja tämä aligraafi on suurin mahdollinen täydellinen aligraafi. Täydellisyysehto on todella vahva: kaikkien klusterien i ja j tietoalkioiden täytyy olla yhdistetty kaarilla ennen kuin klusterit voivat yhdistyä. Kokoavassa hierarkkisessa klusteroinnissa kaikki klusterit lopulta yhdistyvät yhdeksi klusteriksi. Näin ollen ehtoa kaarella yhdistämisestä heikennetään klusteroinnin edetessä. [4]

Sovelletaan kauimmainen naapuri -menetelmää kuvan 4 pistejoukolle. Pistejoukosta saadaan taulukon 3 matriisi. Ensimmäisellä klusterointitasolla yhdistetään kaksi lähintä klusteria, klusterit B ja C. Ne muodostavat klusterin BC. Uusi etäisyysmatriisi on taulukossa 6.

	A	BC	D	E
A	0	1,80	3,21	3,98
BC	1,80	0	1,71	3,70
D	3,21	1,71	0	3,04
E	3,98	3,70	3,04	0

Taulukko 6: Etäisyysmatriisi - klusterointitaso 1

Seuraavaksi määritetään uuden klusterin etäisyys muihin klustereihin. Kauimmainen naapuri -klusteroinnissa etäisyys määräytyy toisistaan kauimmaisten jäsenten välisenä etäisyytenä. Klusterin BC etäisyys klusterista A on 1,80, koska B:n etäisyys A:sta on suurempi kuin C:n. C:n etäisyys A:sta on 1,50. Vastaavasti BC:n etäisyys D:stä on 1,71 ja E:stä 3,70. Näin muodostuva etäisyysmatriisi on taulukossa 7.

	A	BCD	E
A	0	1,80	3,98
BCD	3,20	0	3,70
E	3,98	3,70	0

Taulukko 7: Etäisyysmatriisi - klusterointitaso 2

Klusterin BC etäisyys A:sta kasvaa D:n liitoksen myötä, koska sen etäisyys A:sta on suurempi kuin B:llä ja C:llä. Seuraavaksi yhdistetään klusterit BCD

ja A. Viimeisessä vaiheessa yhdistetään klusterit $ABCD$ ja E. Kokoava klusterointi päättyy, kun kaikki tietoalkiot ovat yhdessä klusterissa.

5.3 Keskiarvomenetelmä

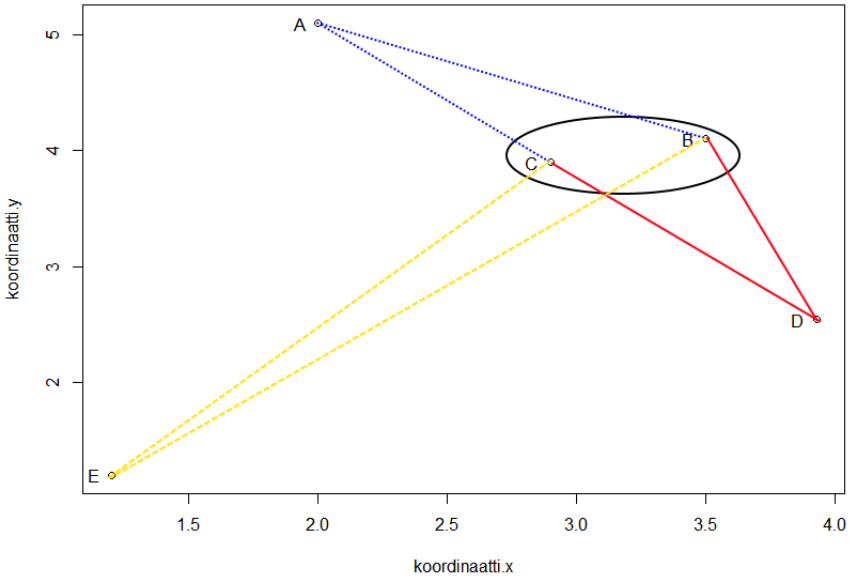
Keskivarvomenetelmässä klusterien väliset etäisyydet lasketaan klusterien jäsenten välisten etäisyyksien keskiarvon perusteella. Etäisyydet lasketaan kaavalla $\frac{1}{n_i+n_j}\sum_{x\in iy\in j} d(x,y)$, missä i ja j ovat klusterit, joiden etäisyyksiä tarkastellaan, n_i ja n_j ovat i:n ja j:n jäsenten lukumäärät ja x ja y ovat klusterien jäseniä. Lopullinen etäisyys siis määräytyy kaikkien i:n ja j:n jäsenten välisten etäisyyksien keskiarvona. Luonnollisesti nämä operaatiot vaativat enemmän laskentaa kuin etäisyyksien laskenta kauimmainen naapuri- ja lähin naapuri -menetelmissä. [2]

Lancen ja Williamsin [5] käyttämien luokittelukritteerien mukaan keskiarvomenetelmä on kombinatorinen ja yhteensopiva. Keskiarvomenetelmä on tilaa muokkaamaton, sillä se ei laajenna tai supistaa tilaa klustereiden ympärillä merkittävästi.

Lähdetään taas liikkeelle kuvan 4 pistejoukosta ja taulukon 3 etäisyysmatriisista. Ensimmäisellä klusterointitasolla yhdistetään kaksi klusteria joiden etäisyys toisistaan on pienin, klusterit B ja C. Seuraavaksi täytyy päivittää klusterin BC etäisyys muihin klustereihin. Kuvassa 5 on havainnollistettu, miten klusterien etäisyydet lasketaan. Esimerkiksi, kun lasketaan klusterin BC etäisyys klusterista A, lasketaan B:n ja C:n etäisyydet A:sta yhteen ja jaetaan lukumäärällä eli aiemmin esitetyn kaavan mukaan $\frac{1.80+1.50}{2} = 1,60$. Näin saadaan taulukossa 8 oleva etäisyysmatriisi.

	A	BC	D	E
A	0	1,60	3,20	3,98
BC	1,60	0	1,66	3,45
D	3,20	1,66	0	3,04
E	3,98	3,45	3,04	0

Taulukko 8: Etäisyysmatriisi - klusterointitaso 1



Kuva 5: Etäisyyksien laskenta

Seuraavaksi yhdistetään klusterit BC ja A. Klusterin ABC etäisyys klusterista D on $\frac{3{,}21+1{,}62+1{,}71}{3} = 2,18$ ja E:stä $\frac{3{,}98+3{,}70+3{,}19}{3} = 3,62$. Yhdistetään klusterit ABC ja D. Klustereiden yhdistymisestä muodostuva matriisi on talukossa 9. Klusterin $ABCD$ etäisyys E:stä on $\frac{3{,}98+3{,}70+3{,}19+3{,}04}{4} = 3,48$.

	ABC	D	E
ABC	0	2,18	3,62
D	2,18	0	3,45
E	3,62	3,45	0

Taulukko 9: Etäisyysmatriisi - klusterointitaso 2

Viimeisellä klusterointitasolla yhdistetään klusterit $ABCD$ ja E. Kokoava klusterointi päättyy, kun kaikki klusteroitavat tietoalkiot ovat samassa klusterissa.

6 Yhteenveto

Hierarkkinen klusterointi on klassinen teoreettisen tietojenkäsittelytieteen aihe. Tutkielman tavoitteena oli antaa lukijalleen kuvaus hierarkkisesta klusteroinnista ja havainnollistaa klusterointiprosessia esimerkkien avulla. Luonnollisesti näin suppeassa tutkielmassa ei voi perinpohjaisesti käsitellä näin monimutkaista aihetta. Käsittelemättä jäivät esimerkiksi geometriset klusterointimenetelmät. Lisätietoa erilaisista klusterointimenetelmistä, tiedonlouhinnasta ja koneoppimisesta yleisesti löytyy tässä tutkielmassa käytetyistä lähteistä.

Viitteet

[1] Ruichu Cai, Zhenjie Zhang, Anthony K. H. Tung, Chenyun Dai, and Zhifeng Hao. A general framework of hierarchical clustering and its applications. *Information Sciences*, 272:29–48, 7/10 2014.

[2] Krzysztof J. Cios, Witold Pedrycz, Roman W. Swiniarski, and Lukasz A. Kurgan. *Data Mining, A Knowledge Discovery Approach*. Springer, 2007.

[3] William H.E. Day and Herbert Edelsbrunner. Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of Classification*, 1(1):7–24, 1984.

[4] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.

[5] G. N. Lance and W. T. Williams. A general theory of classificatory sorting strategies: 1. hierarchical systems. *The Computer Journal*, 9(4):373–380, February 01 1967.

[6] Kari Lehmussaari. Hierkarkkinen klusterointi. *Klusterointimenetelmat-seminaari, Helsingin yliopisto*, 2002.

[7] Tom Michael Mitchell. *The discipline of machine learning*, volume 17. Carnegie Mellon University, School of Computer Science, Machine Learning Department, 2006.

[8] Fionn Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4):354–359, 1983.

[9] F. James Rohlf. Singlelink clustering algorithms. *Handbook of Statistics*, 2:267–284, 1982.

[10] Beverly Park Woolf. *Building Intelligent Interactive Tutors: Student-centered Strategies For Revolutionizing E-learning*. Morgan Kaufmann, 2010.

[11] Osmar R. Zaiane. *Introduction to data mining: Principles of Knowledge Discovery in Databases*, CMPUT609 University of Alberta, Edmonton, 1999.

Pervasiivisten pelien tarjoamat mahdollisuudet opetuskäytössä

Ville Lahtinen

Tiivistelmä.
Nopeasti kehittyvä teknologia on tuonut paljon mahdollisuuksia täysin uudenlaisten pelikokemusten luomiseen, joista hyvänä esimerkkinä toimivat pervasiiviset pelit. Pervasiiviset pelit laajentavat pelimaailman todelliseen maailmaan ja jokapäiväiseen elämäämme, joissa erityisesti mobiiliteknologialla on usein tärkeä rooli.

Pervasiivisten pelien hyödyntämisestä opetuskäytössä on toistaiseksi melko vähän kokemuksia, mutta alustavat tutkimustulokset osoittavat niiden motivoivan oppilaita monesti hyvinkin paljon. Pelien vaikutuksesta oppimistuloksiin on kuitenkin vielä vähän näyttöä.

Avainsanat ja -sanonnat: Pervasiivisuus, sekoitettu todellisuus, oppimispelit.

1. Johdanto

Pelien käyttö yhtenä osana opetusta on viime vuosina herättänyt kasvavaa kiinnostusta tutkijoissa, opettajissa ja varmasti erityisesti lapsissa ja nuorissa – vietäväthän nuoret nykyisin valtavasti aikaa erilaisten digitaalisten pelien parissa. Tuoreimman lasten mediabarometrin [Suoninen, 2014] mukaan 50 prosenttia 7–8-vuotiaista lapsista pelaa digitaalisia pelejä ainakin kerran päivässä tai melkein joka päivä, ja 84 prosenttia ainakin kerran viikossa. Pelaaminen myös aloitetaan nykyisin nuorempana kuin aiemmin.

Pelkästään tietokoneen tai tabletin ruudulla tapahtuvia perinteisiä oppimispelejä on tutkittu melko paljon, tosin yksiselitteisiä tuloksia niiden vaikutuksesta oppilaiden oppimiseen ei vielä ole saatu. Useat tutkimukset ovat kuitenkin nähneet pedagogisesti hyvin suunnitelluissa oppimispeleissä potentiaalia oppimistulosten parantamisessa, kunhan ne sisältävät ominaisuuksia, jotka johdattavat käsittelemään opetettavaa aihetta aktiivisesti ja monesta näkökulmasta, ja kun ne tarjoavat tarpeeksi älykästä palautetta pelaajille [Erhel and Jamet, 2013]. Myös se, kuinka olennainen osa opetettava aihe on itse pelimaailmaa ja -mekaniikkaa, vaikuttaa usein huomattavasti oppimistuloksiin: pelit, joissa oppiminen on selkeästi irrallinen osa pelikokemuksesta, kuten esimerkiksi yksittäisten laskutehtävien ratkaisu pelissä etenemisen edellytyksenä, eivät motivoi oppilaita yhtä paljon, kuin jos laskutehtävät ovat olennainen osa vaikkapa taistelukohtauksia [Habgood and Ainsworth, 2011].

Kaikki digitaalinen pelaaminen ei kuitenkaan ole rajoittunut pelkästään näytön äärellä tapahtuvaksi kokemukseksi. Kun pelimaailma laajenee todelliseen maailmaan, puhutaan *pervasiivisista peleistä* (*pervasive games*). Muita läheisiä käsitteitä ovat muun muassa *sekoitetun todellisuuden pelit* (*mixed/hybrid reality games*), *jokapaikan pelit* (*ubiquitous games*), *lisätyn todellisuuden pelit* (*augmented games*) ja *sijaintiin perustuvat pelit* (*location-based games*). Kaikkien termien suomennokset eivät varmasti ole mitenkään vakiintuneita, ja muutenkin eri termejä saatetaan käyttää eri lähteissä tarkoittaen hieman eri asioita. En tässä tutkielmassa kuitenkaan aio kiinnittää termien hienoisiin merkityseroihin kovinkaan paljon huomiota, vaan käytän termiä pervasiiviset pelit eräänlaisena kattokäsitteenä, joka sulkee sisäänsä suurimman osan muista läheisistä käsitteistä.

Matkapuhelimien yleistyttyä vuosituhannen alussa uutta teknologiaa hyödyntävät pervasiiviset pelit alkoivat saavuttaa suosiota, ja älypuhelinten löydettyä tiensä markkinoille nämä pelit tulivat entistäkin helpommin kaikkien saataville. Genreen voidaan laskea mukaan esimerkiksi erilaisia aarteenmetsästys-, taistelu-, seikkailu- ja live–roolipelejä. Yhteistä kaikille näille on, että tavallisen arjen ja pelimaailman rajat hämärtyvät jollain tasolla: fyysisesti, ajallisesti tai sosiaalisesti. Esimerkiksi pelilautana voi toimia vaikka kokonainen kaupunki tai koko maailma, mistä syystä pelien tarina ja mekaniikka saattaa usein muodostua erittäin moniulotteiseksi. [Stenros *et al.*, 2012.]

Tässä tutkielmassa teen kirjallisuuskatsauksen digitaalista tekniikkaa sisältävien pervasiivisten pelien hyödyntämisestä opetuskäytössä. Tutkielmassa selvitän, miten pervasiivisuutta on käytetty hyväksi oppimispelien suunnittelussa ja toteutuksessa, millaisia oppimistuloksia näillä peleillä on saatu aikaiseksi, sekä pohdin miten ne mahdollisesti voisivat tuoda uudenlaista yhteisöllisyyden kulttuuria ja parempaa oppimista kouluihin.

2. Pelit, pelaaminen, opetus ja oppiminen

Johan Huizingan [1955] määritelmän mukaan pelaaminen on vapaaehtoista ja itsetarkoituksellista, selkeästi muusta elämästä erillistä toimintaa, jota sitovat tietyt säännöt. Tähän määritelmään on myöhemmin lisätty muun muassa vaatimukset siitä, että kaikki pelaajat tietävät pelin olevan fiktiivistä, sekä että pelin lopputulos ei ole etukäteen selvä [Caillois and Barash, 1961]. Näiden määritelmien pohjalta on syntynyt termi *pelin taikapiiri* (*magic circle*), joka kuvaa pelin ja todellisuuden erottavaa rajaa [Hinske *et al.*, 2007]. Määritelmän mukaan taikapiirillä on pelialueen muodostavat fyysiset rajat, pelin alkamisen ja loppumisen määrittävät ajalliset rajat, sekä pelin roolit ja säännöt määrittävät sosiaaliset rajat. Taikapiirin voi nähdä eräänlaiseksi sosiaaliseksi sopimukseksi pelaajien välillä, jonka sisällä ollessaan kaikki tietävät peliin liittyvien sääntöjen pätevän. Taika-

piiriä terminä on myös kritisoitu [mm. Malaby, 2007], mutta käsiteltäessä pervasiivisia pelejä, koen sen kuitenkin havainnolliseksi: kun vähintään yksi mainituista taikapiirin rajoista rikotaan, voidaan peliä kutsua pervasiiviseksi. Pelimaailma ei siis tällöin enää ole selkeästi muusta todellisuudesta erottuva alue, vaan se on jollain tasolla tunkeutunut arkimaailmaan.

Kuten useiden käsitteiden, myös pelaamisen määritelmä elää ja muuttuu yhteiskunnan mukana. Digitaalisia pelejä ei enää nähdä pelkkänä lasten ajanvietteenä, sillä pelaajien keski-ikä nousee jatkuvasti, ollen nyt noin 35 vuotta [Entertainment Software Association, 2015]. Pelin ja arjen rajojen hämärtyessä on alettu siirtyä eräänlaiseen *pelillisyyden* (*gamefulness*) kulttuuriin. Yhden näkökulman mukaan pelaamisesta ja pelillisyydestä on tullut niin keskeinen osa useilla yhteiskuntamme aloilla, että voidaan jo puhua *pelillisestä* tai *leikillisestä yhteiskunnasta* (*ludic society*) [Stenros *et al.*, 2007]. On toki monia mielipiteitä siitä, onko tämä kehitys tavoiteltavaa vai ei, mutta trendi on kuitenkin olemassa ja se on nähtävissä myös kouluissa.

Useat tutkimukset [mm. Bateson and Martin, 2013] ovat osoittaneet *leikillisyyden* (*playfulness*) yhteyden muun muassa parempaan elämänlaatuun ja parempaan oppimiseen. Pedagogisessa viitekehyksessä tarkasteltuna leikillisyydestä voidaan puhua myös *leikillisenä oppimisena* (*playful learning*). Leikillisyys ja pelillisyys ovat käsitteinä läheisiä ja toisiaan täydentäviä: Deterdingin *et al.* [2011] mukaan leikillisyys on näistä kahdesta laajempi käsite.

Vaikka opetuksen pelillistäminen ja oppimispelit eivät suinkaan ole mitään uusia käsitteitä, on tietokoneiden ja mobiililaitteiden räjähdysmäinen yleistyminen kuitenkin luonut erityisesti digitaalisille oppimispeleille valtavan suuret markkinat, ja niille on myös asetettu suuria ja osin ehkä epärealistisiakin odotuksia opetuksen ja oppimisprosessien mullistamisesta.

2.1. Pervasiiviset pelit

Pervasiiviset pelit siis haastavat perinteisen pelin taikapiirin. Pisimmälle vietynä pervasiivisuus voi tarkoittaa kaikkien rajojen katoamista pelimaailmasta, niin etteivät edes pelaajat välttämättä enää tiedä mitkä elementit ovat osa peliä ja mitkä eivät. Periaatteessa kaikki pelaajien kokemat aistihavainnot, muut ihmiset sekä ympäristön tapahtumat voivat siis olla osa pelin juonta, tai sitten eivät. Tätä piirrettä onkin kritisoitu muun muassa siitä, että täysin sivulliset ihmiset saattavat tällöin joutua keskelle pelin tapahtumia sitä ollenkaan haluamatta. [Niemi *et al.*, 2005.]

Taikapiirin fyysisten rajojen rikkoutuessa muita haasteita saattavat tuottaa pelin mahdollisesti aiheuttamat vaaratilanteet liikenteessä, tai pelien laajentuminen alueille, joissa ne tuottavat häiriöitä yleiselle järjestykselle, kuten sairaalat ja lentokentät. Kun pelin taikapiirin ajalliset rajat hämärtyvät, voi peli käytännössä

olla käynnissä jatkuvasti. Peli saattaa siis ilmoittaa itsestään milloin tahansa, eikä ääriesimerkissä pelaaja tällöinkään välttämättä voi olla täysin varma, kuuluuko jokin arjen tapahtuma peliin vai ei. Taikapiirin sosiaalisten rajojen rikkoutuessa pelaajien ja sivullisten ihmisten roolit hämärtyvät, minkä seurauksena sivullisten henkilöiden toiminta voi vaikuttaa pelin etenemiseen eri tavoin. Yhtenä myönteisenä mahdollisuutena tässä voi nähdä uusien sosiaalisten kontaktien luomisen, mikäli peli siis sisältää kontaktin ottamista muihin pelaajiin tai sivullisiin. Mutta kuten todettua, sivullisten suhtautumista siihen, että heistä tulee osa peliä pyytämättään, ei mitenkään voi arvioida etukäteen. Sosiaalisten rajojen rikkominen onkin yksi pervasiivisten pelien eniten eettisiä ja moraalisia kysymyksiä herättänyt piirre. [Montola, 2005.] Pelisuunnittelijat eivät voi mitenkään ennustaa pervasiivisten pelien kaikkia tapahtumia etukäteen, mutta mahdolliset vaaratilanteet ja riskitekijät tulisi ehdottomasti kartoittaa jo suunnitteluvaiheessa. Hyvällä suunnittelulla pelaajille ja ympäristölle muodostuvia riskejä voidaan varmasti vähentää.

Stenros *et al.* [2007] tekevät jaottelun leikillisen ja vakavamielisen ajattelutavan, sekä arkisen ja leikillisen kontekstin välille, joiden avulla erilaisia aktiviteetteja voidaan määritellä. Perinteisesti pelit ja arkipäiväiset askareet on pystytty sijoittamaan melko helposti syntyvään nelikenttään (ks. taulukko 1). Pervasiiviset pelit kuitenkin sijaitsevat usein eräänlaisella harmaalla alueella eri lokeroiden välillä, eli pelikokemuksen edetessä ajattelutavat ja kontekstit voivat vaihdella hyvinkin paljon tilanteesta riippuen.

		Ajattelutapa: Leikillinen	Ajattelutapa: Vakava
Konteksti	Leikillinen	Lasten leikit, pelit	Ammattimainen urheilu
Konteksti	Arkinen	Rullalautailu, parkour	Jokapäiväiset rutiinit

Taulukko 1. Esimerkkejä aktiviteeteista jaoteltuna ajattelutavan ja kontekstin mukaan [Stenros *et al.*, 2007]. Suomennos oma.

2.2. Pelit opetuskäytössä

Lapset ja nuoret pelaavat paljon, ja on toki hyvä jos pelaaminen edesauttaa myös uusien asioiden oppimista. Teknologian ja pelien hyödyntäminen opetuksessa

kouluissa herättää kuitenkin tunteita puolesta ja vastaan. Todd Oppenheimerin [2007] mukaan koulujen uusiin laitteisiin käyttämät rahat ovat suoraan pois opetuksen perusteista, minkä lisäksi opetuksen digitalisoituminen myös rapauttaa tehokkaasti oppilaiden kyvyn pohtia asioita, kuunnella, tuntea empatiaa ja olla luovia. Neil Postman taas kritisoi kirjoissaan [mm. 1985 ja 1995] paljon amerikkalaista opetusjärjestelmää, syyttäen tätä humanististen päämäärien unohtamisesta, opetuksen suuntautuessa yhä enemmän kasvattamaan oppilaista pelkästään tuottavaa työvoimaa markkinoiden ja teollisuuden tarpeisiin. Tietokoneiden tuloa kouluihin hän vastusti, koska se tulisi väistämättä johtamaan yhteisöllisyyden radikaaliin vähenemiseen. Myös yhteiskunnan viihteellistymisen seurauksista hän teki synkkiä ennusteita, väittäen sen johtavan yhteiskunnallisen ja poliittisen keskustelun pinnallistumiseen.

Vaikka Postman kirjoitti kirjansa vuosikymmeniä sitten, ovat hänen ajatuksensa edelleen hyvin ajankohtaisia tänäkin päivänä. On syytä miettiä, mitä tavoitteita koulutuksella ylipäätään halutaan saavuttaa, ja miten opetusteknologia ja oppimispelit tukevat näitä tavoitteita. Oppimispelien hyötyjä voidaan toki tutkia suhteessa siihen, miten hyvin ne auttavat uusien asioiden oppimisessa ja oppilaiden motivaation parantamisessa. Mutta kun koulutuksen yhtenä tavoitteena nähdään myös oppilaiden kasvattaminen yhteiskunnan täysivaltaisiksi jäseniksi, mihin kuuluvat olennaisena osana hyvät sosiaaliset taidot, kyky yhteistyöhön sekä oman paikan löytäminen osana ympäröivää yhteisöä, asettavat nämä oppimispeleille aivan uusia haasteita.

En tässä tutkielmassa käsittele perinteisiä yksin tietokoneella pelattavia pelejä, mutta varmasti nekin voivat osaltaan tarjota vastauksia näihin haasteisiin. Näen kuitenkin pervasiivisissa peleissä suuren potentiaalin tuoda uudenlaista yhteisöllisyyden kulttuuria kouluihin, jossa oppiminen tapahtuu yhdessä toimimalla, pelaten ja leikkien, teknologiaa hyödyntäen, mutta vain silloin kun se tuo opetustapahtumaan jotain lisäarvoa. Parhaassa tapauksessa oppiminen siirtyy tällöin myös koulun ulkopuolelle: fyysisesti, ajallisesti ja sosiaalisesti.

2.3. Kohti oppijakeskeistä oppimista

Perinteisesti kouluopetus on rakentunut opettajavetoisen opettamisen ympärille, joka pohjautuu paljon *behavioristiseen oppimiskäsitykseen*. Tämän oppimiskäsityksen keskeisiä piirteitä ovat oppilaan passiivinen rooli oppimisprosessissa, opetuksen näkeminen tiedon siirtämisenä opettajalta oppilaalle sekä usein irrallisten ja yksittäisten tietojen ja taitojen ulkoa opettelu [Sahlberg ja Leppilampi, 1994]. Opettajavetoisella opettamisella on oma paikkansa koulussa, eikä se varmasti ole häviämässä mihinkään ainakaan lähiaikoina. Kuitenkin tämän opetustavan on todettu johtavan usein ainoastaan pinnalliseen oppimiseen sekä heikkoon oppilaiden motivointiin ja innostamiseen [Bransford *et al.*, 1999].

Osittain näistä syistä erilaiset oppijakeskeiset opetusmenetelmät ovat alkaneet nousta perinteisemmän opetustapahtuman rinnalle. Näissä menetelmissä oppilaille annetaan suurempi vastuu oppimisesta, ja ne vaativat oppilailta enemmän uuden tiedon aktiivista prosessointia. Oppijakeskeisten menetelmien taustalla vaikuttavat paljon *kognitiivinen* sekä *konstruktivistinen oppimiskäsitys*. Kognitiivinen oppimiskäsitys näkee oppimisen tiedon prosessointina: oppiminen synnyttää jäsentyneitä ajatuksia ja periaatteita, joista muodostuu toimintaa ohjaavia sisäisiä rakenteita ja malleja, eli skeemoja. Uuden tiedon omaksuminen nähdään olevan aina riippuvaista aiemmasta tiedosta. Konstruktivistisen oppimiskäsityksen mukaan tieto ei siirry, vaan oppija rakentaa sen itse uudelleen. Tieto on aina yksilöiden ja yhteisöjen sosiaalisesti rakentamaa, ja ymmärtäminen ja ajattelu nähdään keskeisenä osana oppimisprosessia. [Sahlberg ja Leppilampi, 1994.]

Digitaalista tekniikkaa hyödyntävän opetuksen voi nähdä pohjautuvan myös *konnektivistiseen lähestymistapaan*, jonka mukaan oppiminen on kytkentöjen ja yhteyksien luomista eri tietolähteistä ja verkostoista saadun tiedon pohjalta: tietolähteinä voivat toimia esimerkiksi keskustelufoorumit ja sosiaalinen media [Bell, 2010]. Myös *design-suuntautunut pedagogiikka* korostaa teknologian hyödyntämistä opetuksessa: se pyrkii kehittämään erityisesti tulevaisuuden yhteiskunnan kansalaisilta vaadittuja taitoja, joihin luetellaan esimerkiksi luovuus ja innovatiivisuus, kriittinen ajattelu sekä ongelmanratkaisukyvyt [Vartiainen *et al.*, 2012].

Humanistiseen psykologiaan pohjautuva *kokemuksellinen oppiminen* ottaa tavoitteekseen oppilaiden minän kasvun sekä itsensä toteuttamisen [Sahlberg ja Leppilampi, 1994]. Erityisesti näiden tavoitteiden toteutumista nykykoulutuksessa pohtimalla ollaankin jo aiemmin mainitun Postmanin kritiikin ytimessä. Koska oppiminen oppimispeleillä tapahtuu vahvasti oppijakeskeisesti, voi oppimispelejä suunniteltaessa tapauksesta riippuen hyödyntää useita edellä mainittuja oppimiskäsityksiä.

2.4. Pervasiivinen oppiminen

Kun käsitellään pervasiivisia oppimispelejä, on hyvä määritellä myös mitä tarkoitetaan *pervasiivisella oppimisella*. Yhden määritelmän [Thomas, 2006] mukaan pervasiivinen oppiminen on sosiaalinen prosessi, joka yhdistää oppijat laitteiden, pelaajien ja tapahtumien yhteisöön. Tämä yhteisö opettaa heitä, samalla kun he itse opettavat ja antavat oman panoksensa yhteisölle. Pervasiivinen oppiminen on myös autonomista, eli vastuu oppimisesta on oppijalla itsellään, eikä yksittäistä auktoriteettihahmoa välttämättä ole antamassa kysymyksiä tai vastauksia. Vastauksia saattaakin olla lähteestä riippuen useita, eikä joissain tapauksissa mikään niistä ole toista oikeampi tai parempi. Oppiminen tapahtuu eri aikoina ja eri paikoissa, periaatteessa kaikki tilanteet voivat edistää oppimista. Oppijat myös rakentavat merkityksellisiä oppimistilanteita omissa ympäristöissään,

mikä auttaa heitä paremmin ymmärtämään opittujen asioiden merkitykset ja täten myös hyödyntämään niitä tehokkaammin omissa elämissään. Sosiaalinen kanssakäyminen nähdään usein tärkeänä osana pervasiivista oppimista. Tässä taustalla vaikuttaa paljon *sosiaalisen kehityksen teoria*, jonka mukaan yksilön tietoisuus rakentuu sosiaalisissa suhteissa, yhteiskunnan kulttuurin pohjalta, ja tiedollisen kehityksen päämääränä on tällöin yhteisöön sosiaalistuminen [Daniels, 2005].

Kun keskitytään erityisesti kouluissa pelattaviin oppimispeleihin, tulee toki huomioida, etteivät kaikki edellä mainitut ominaisuudet välttämättä toteudu. Kouluissa opettajilla on edelleen auktoriteettiasema oppilaisiin nähden ja pelitapahtumilla on usein rajattu aika ja tapahtuma-alue. Ideaalitilanne toki on, että hyvin suunnitellut oppimispelit kannustavat ja motivoivat oppilaita pelaamaan ja oppimaan myös koulun ulkopuolella. Kun tämä toteutuu, voi oppimisen nähdä olevan aidosti pervasiivista.

2.5. Pervasiivisten oppimispelien erityispiirteitä

Yksi tapa jaotella pelejä on niiden tavoitteen mukaan, eli onko ne suunniteltu pelkästään viihdyttämään, pelkästään opettamaan vai jotain tältä väliltä [Avouris and Yiannoutsou, 2012]. En tässä tutkielmassa käsittele ei-opetuksellisia pelejä, mutta rajanveto näiden ja niin sanottujen hybridipelien välillä ei aina ole kovin yksinkertaista: täysin viihteelliseksi suunniteltu peli voi joissain tapauksissa opettaa jopa enemmän kuin epäonnistunut oppimispeli. Esimerkiksi uusien sosiaalisten kontaktien luontiin kannustavat viihteelliset pervasiiviset pelit voivat omalla tavallaan olla opetuksellisesti arvokkaita. Kaikki tässä tutkielmassa myöhemmin mainitut pelit sisältävät kuitenkin selkeän pedagogisen tavoitteen.

Ängeslevä [2014] tekee jaottelun perinteisten oppimispelien ja niin sanottujen hiekkalaatikkopelien välille. Hiekkalaatikkopelien ominaisuuksia ovat avoin pelimaailma ilman ennalta määrättyjä tavoitteita sekä sääntöjen löyhyys. Näissä usein sekä opettajat että oppilaat voivat tuottaa ja muokata pelin sisältöjä. Verrattuna valmiiseen pelisuunnittelijoiden alusta loppuun rakentamaan peliin, vaatii tämä toki opettajilta aktiivisempaa otetta jo ennen varsinaisen opetusprosessin alkamista. Pelin taikapiirin fyysisten rajojen rikkoutuessa hiekkalaatikkopeleissä on opettajilla vielä korostetusti suurempi vastuu pitää huolta oppilaiden turvallisuudesta ja lakien noudattamisesta.

Ejsing-Duunin *et al.* [2013] mukaan pervasiivisiin oppimispeleihin osallistuvat pelaajat vuorottelevat jatkuvasti pelin omien tavoitteiden sekä oppimistavoitteiden välillä. Pelimaailman ja reaalimaailman sekoittuessa pelaajat joutuvat myös tarkkailemaan peliä monien erilaisten kehysten läpi. Erityisesti tämä jat-

kuva tavoitteiden ja erilaisten kehysten välillä siirtyminen voivat tuottaa kehittäviä oppimisprosesseja, sillä pelaajat joutuvat käyttämään luovuuttaan uusien ennustamattomien pelitilanteiden syntyessä. Näissä peleissä huijaamisen estäminen voi osoittautua haastavaksi, juuri tapahtumien vaikeasta ennustettavuudesta johtuen. Huijaaminen voi toki muodostua ongelmaksi ainoastaan pelin tavoitteiden saavuttamista ajatellen: jos oppimistavoitteet saavutetaan sääntöjä kiertäen, ei tämä suinkaan ole huijaamista, vaan tehokkaampaa oppimista.

3. Tutkimuksia oppimistuloksista

Opetuskäyttöön suunniteltuja pervasiivisia pelejä on viimeisen vuosikymmenen aikana julkaistu jonkin verran, mutta näissä tapahtuvaa oppimista on tutkittu yllättävän vähän. Sen sijaan tutkimukset ovat keskittyneet enemmän metodin oppilaita motivoiviin vaikutuksiin, ja tällä saralla tulokset ovat keskimäärin olleet hyvin lupaavia. Pervasiiviset oppimispelit johtavat usein pelaajien syvään pelikokemukseen uppoutumiseen, eli niin sanottuun flow-tilaan, jossa tietoisuus ympäröivästä maailmasta vähenee huomattavasti. [Huizenga *et al.*, 2009.] Motivoituneet oppijat ovat innostuneita ja keskittyneitä, ja motivaation ja sisäsyntyisen oppimisen välillä on esitetty olevan yhteys [Schwabe and Göth, 2005]. Kuitenkaan pelkästään oppilaiden kasvanut motivaatio ei Schwaben ja Göthinkään mielestä riitä johtopäätöksien vetämiseen oppimistulosten paranemisesta.

Tässä osiossa käsittelen eri tutkimuksissa saatuja tuloksia peli kerrallaan.

3.1. Digital Economy

Digital Economy -pelissä opiskelijoille annetaan sijainti, jossa on opetettavaan aiheeseen liittyvä ongelma, joka oppilaiden tulee tunnistaa. Ongelman tunnistettuaan oppilaiden tulee määritellä ongelma ja lähettää tästä kuvaus pelinjohtajalle arvioitavaksi. Lopuksi oppilaiden tulee keksiä toimiva ratkaisu ongelmaan. Näitä yksinkertaisia sääntöjä soveltamalla opettajien tehtäväksi jää ainoastaan erilaisten ongelmien keksiminen ja vastausten arviointi, eli se on siis tietyllä tapaa eräänlainen hiekkalaatikkopeli. Pelistä tässä käytetty nimi Digital Economy viittaa digitaalisen talouden kurssiin, jota varten se alun perin suunniteltiin, mutta konseptia voi hyödyntää myös lukuisten muiden eri oppiaineiden ja aihepiirien käsittelyssä. Peli suunniteltiin aikaan ennen mobiilisovellusten yleistymistä, joten teksti- ja multimediaviestit toimivat tiedonvaihtokanavana pelaajien ja pelinjohtajan välillä. Joukkueiden arvostelu tapahtuu toimitettujen ratkaisujen innovatiivisuuden ja nopeuden perusteella. Joukkueiden keräämät pisteet voidaan myös julkaista internetissä, mikä lisää peliin kilpailullisen elementin. [Kittl and Petrovic, 2008.] Peli rikkoo siis taikapiirin fyysiset rajat, mutta ajallisesti sen tapahtumat on kuitenkin määrätty ennalta ja opettajalla on merkittävä rooli pelin kulussa.

Pelillä saavutettuja oppimistuloksia verrattiin perinteiseen opettajavetoiseen opetukseen monen muuttujan varianssianalyysin perusteella (MANOVA). Tutkimusta varten digitaalista taloutta käsittelevän yliopistokurssin opiskelijat jaettiin kahteen sadan hengen ryhmään, joista toista ryhmää opetettiin perinteisen tavan mukaan, kun taas toinen ryhmä opetteli samat asiat Digital Economy -pelin avulla. Tulokset osoittavat pelin aktivoineen oppilaita huomattavasti verrokkiryhmää enemmän. Peliä pelanneiden keskuudessa koettiin enemmän ilon tunnetta, käsiteltyihin oppimateriaaleihin suhtauduttiin paljon myönteisemmin ja oppimistulokset paranivat. [Kittl and Petrovic, 2008.] Toki pelin uutuudenviehätys saattoi osaltaan vaikuttaa tuloksiin, mutta tästä huolimatta tulokset antavat tukea pervasiivisten pelien hyödyntämiselle täydentävänä lisänä perinteisen opetuksen rinnalla.

3.2. SmartFeet

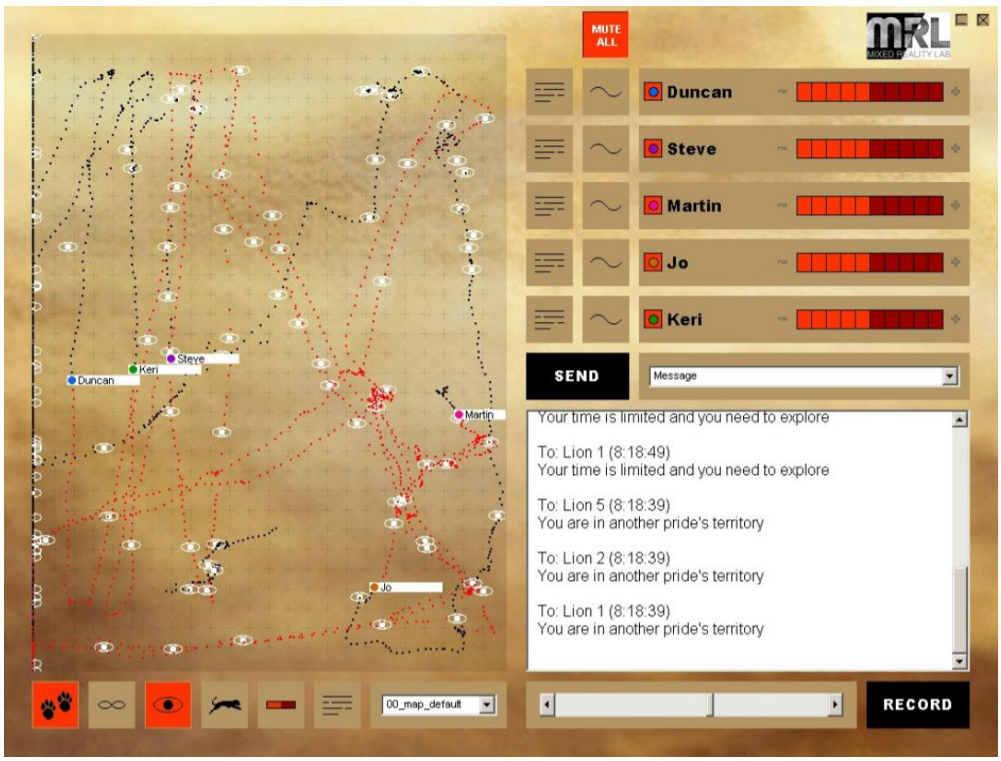
SmartFeet itsessään ei oikeastaan ole peli, vaan alusta, jolle opettajat, kasvattajat ja myös oppilaat voivat helposti luoda omia pervasiivisia pelejään. Näin se muistuttaa paljon Digital Economyn pohjana olevaa konseptia, ollen eräällä tapaa modernimpi versio siitä: merkittävimpinä eroina on kehittyneemmän teknologian hyödyntäminen sekä monipuolisempien toiminnallisuuksien mahdollistaminen. Peliympäristönä tässäkin toimii yleensä todellinen maailma, ja virtuaalinen pelialusta kulkee mukana pelaajien mobiililaitteissa. Vaihtoehtoisesti peli voi olla myös kokonaan verkossa, jolloin pelaajat pelaavat luokkatilassa. [Ängeslevä, 2014.] Tällöin tosin peliä ei välttämättä enää voi luokitella pervasiiviseksi, jos siis yhtäkään pelin taikapiirin rajoista ei rikota. Pelit voivat sisältää tarinankerrontaa ja myös pelaajien keskinäinen kommunikointi voidaan mahdollistaa. Tehtävänannot voivat olla periaatteessa mitä tahansa: kaikkea aina luovasta kirjoittamisesta nauhoitettaviin taideperformansseihin. [Viita ja Alkio, 2014.]

Yhdessä tutkimuksessa noin kaksikymmentä lukiolaista pelasi SmartFeet-peliympäristöön luotua Smart City -peliä. Pelin teemana oli ekologisesti ja sosiaalisesti kestävä tulevaisuuden Helsinki. Mielenkiintoisena erikoisuutena opiskelijat loivat itse kaikki pelin tehtävät sekä arviointikriteerit, joita noudattamalla opettajat jakaisivat pisteet. Yli puolet opiskelijoista koki pelipäivän erittäin positiiviseksi. Suurin osa koki myös ymmärryksen ja kiinnostuksen käsiteltyyn aiheeseen kasvaneen. Tekniikkaa oli lähes kaikkien mielestä helppo käyttää. [Viita ja Alkio, 2014.] Oppimistuloksia ei kuitenkaan verrattu mihinkään, joten näiden paranemisen osalta johtopäätöksiä ei voida vetää.

SmartFeetin kaltaiset avoimet peliympäristöt antavat perinteisiin peleihin verrattuna suuremman vallan opettajille, mutta ne myös vaativat näiltä paljon viitseliäisyyttä ja suunnittelua – jos ei suunnittelua sitten ulkoisteta kokonaan oppilaille.

3.3. Savannah

Savannah on peli, joka opettaa pelaajille leijonien käyttäytymistä. Pelaajat ovat leijonia savannia symboloivalla pelikentällä, jossa he suorittavat erilaisia tehtäviä, jotka voivat olla esimerkiksi reviirin merkintää tai ruoan hankintaa. Mukana kannettava mobiililaite ilmoittaa ympäristössä mahdollisesti vaanivista uhista, muista leijonalaumoista tai saaliseläimistä. Pelinjohtaja pystyy seuraamaan pelitapahtumien etenemistä reaaliaikaisesti (ks. kuva 1). Tehtäviä suorittaessaan pelaajat oppivat leijonista ja savannien elämästä uniikilla tavalla. [Benford *et al.*, 2004.]



Kuva 1. Pelinjohtajan näkymä Savannahissa. Vasemmalla kartassa näkyvät pelaajien liikkeet ja ylhäällä oikealla näiden energiatasot. Oikean alanurkan tekstikenttää käytetään pelaajien kanssa kommunikointiin. [Benford et al., 2004.]

Tutkimukseen osallistui muutama viiden lapsen ryhmä, jotka pelasivat eri tehtäviä Savannahilla, jonka jälkeen suoritettiin haastattelu pelikokemuksesta. Kokemusta kuvailtiin yleisimmin hyvin lumoavaksi, pelaajat identifioituivat vahvasti leijonien rooliin ja savanniympäristö oli todentuntuinen. Opettajien mukaan peliä pelanneet oppilaat myös osoittivat olevansa vahvasti motivoituneita oppimaan lisää leijonista. Ongelmaksi muodostuivat kuitenkin pelimekaniikan rajoitukset: pelin suunnittelu ja tekniikan hyödyntäminen ei ollut oppimista ajatellen paras mahdollinen, mistä syystä pelaajat keskittyivät usein liikaa oppimisen kannalta väärien asioiden suorittamiseen, eli pelin tavoitteet nousivat pelaajien keskuudessa oppimistavoitteita tärkeämmiksi. [Facer *et al.*, 2004.]

Savannah on suunniteltu alusta asti pelkästään yhden hyvin rajatun aihealueen opettamiseen. Näin se eroaa esimerkiksi aiemmin mainituista Digital Economysta ja SmartFeetistä. Yhteen aiheeseen keskittyvä peli on mahdollista kustomoida huomattavasti yksityiskohtaisemmin graafista ulkoasua myöten, mutta pelaamisen jatkaminen ei välttämättä ole opetuksellisesti mielekästä enää kun käsitellyt asiat on opittu.

3.4. Nuclear Mayhem

Nuclear Mayhem on pelin taikapiirin kaikki rajat rikkova oppimispeli, joka luotiin tukemaan web-teknologioiden yliopisto-opintoja. Peli linkittyy vahvasti yhden kurssin opintosuunnitelmaan, ja pelissä pärjätäkseen opintosuunnitelmassa mainitut asiat tulee hallita hyvin. Peli on käynnissä koko kurssin keston ajan ja se kytkeytyy opiskelijoiden arkielämään monin tavoin: vihjeitä ja tehtäviä voi löytää sosiaalisesta mediasta, niitä voi saada tekstiviestein keskellä yötä tai löytää eri paikoista kaupungissa, kuten kauppojen ikkunoista. Salaisia viestejä voi olla piilotettuna myös muiden kurssien luennoille tai laajemmin eri nettisivuille, jotka opiskelijoiden on löydettävä. Peli on olennainen lisä luennoilla käsitellyille asioille ja sen suorittaminen on ehtona lopputenttiin osallistumiselle. [Pløhn, 2014.]

Tutkimuksessa käsitellyn pelin tarina otti vaikutteita tosielämän tapahtumista ja monia pelin aikana tapahtuneita uutisia hyödynnettiin tarinan edetessä: kurssin aikana laajasti uutisoitu Iranin ydinohjelma otettiin pelin taustatarinaksi. Näin pelistä saatiin entistä pervasiivisempi ja pelin ja todellisuuden raja hämärrytettiin. Kyseessä oli siis *todellisuuden hakkerointi* (*reality hacking*) [Waern *et al.*, 2009].

Tutkimuksessa saatiin selville, että valtaosa peliin kirjautumisista tuli muulloin kuin kurssin opetustilanteiden aikana ja kirjautumisia tapahtui lähes kaikkina kellonaikoina, eli opetuksen ajallinen laajentaminen onnistui. Kaikki kurssille osallistuneet 17 opiskelijaa pitivät tositapahtumien mukaan ottamista peliin myönteisenä asiana ja kaikki pitivät ainakin osia pelistä hauskana ja motivoivana. [Pløhn, 2014.] Nämä tulokset antavatkin tukea pervasiivisten pelien potentiaalille myös opetuksen laajentamisessa perinteisten opetustilanteiden ulkopuolelle. Nuclear Mayhemin melko yksinkertaista konseptia on lisäksi helppo soveltaa eri kouluasteille ja eri oppiaineiden opetukseen, tosin toimivan ja uskottavan tarinan rakentaminen vaatii aina paljon aikaa.

3.5. Frequency 1550

Frequency 1550 on suunniteltu pelattavaksi yhden koulupäivän aikana Amsterdamin keskustassa. Peli muuttaa kaupungin keskiaikaiseksi: pelaajille arvotaan rooliksi joko kerjäläinen tai kauppias, ja pelaajien tavoitteena on erilaisia tehtäviä

suorittamalla kerätä tarpeeksi pisteitä kaupungin kansalaisuuden saavuttamiseksi. Keskiaikaista tunnelmaa luodaan multimodaalisesti muun muassa video- ja audiotallenteiden avulla. Oppilaat liikkuvat ryhmissä, ja tehtävät voivat olla esimerkiksi kysymyksiin vastaamista tai erilaisten luovien ratkaisujen löytämistä esitettyihin ongelmiin. Opetuksellisena tavoitteena pelissä on kiinnostuksen lisääminen historiaan ja erityisesti keskiaikaan. [Admiraal *et al.*, 2011.]

Yhdessä tutkimuksessa 468 toisen asteen opiskelijasta noin puolet osallistuivat peliin ja puolet opettelivat samat asiat perinteisen luento-opetuksen parissa. Kuten ehkä odotettua, peliä pelanneet pitivät kokemusta enimmäkseen mukavana ja motivoivana. Oppimistuloksissa oli suuret erot: peliin osallistuneiden tiedot keskiaikaisesta Amsterdamista olivat huomattavasti paremmat kuin verrokkiryhmällä. Motivaatiossa historiaan oppiaineena tai erityisesti keskiaikaan ei kuitenkaan syntynyt merkittäviä eroja ryhmien välillä. [Huizenga *et al.*, 2009.]

Pelkästään yhden päivän kestävä peli ei tietenkään voi tarjota yhtä syvällistä kokemusta kuin esimerkiksi kokonaisen kurssin ajan käynnissä oleva peli, kuten aiemmin mainittu Nuclear Mayhem, mutta koska oppimistulokset vaikuttavat tämän pelin osalta rohkaisevilta, tarjoaa se oivallisen esimerkin uudenlaisten tehokkaiden oppimismahdollisuuksien luomisesta vaikkapa lyhyillä luokkaretkillä. Savannahin tapaan Frequency 1550 on suunniteltu yhden aiheen opettamiseen, mutta tämän lisäksi myös sen pelialue on rajattu ainoastaan yhteen kaupunkiin, eli se on kaikista käsitellyistä peleistä selkeästi vähiten kustomoituva.

4. Pohdintaa

Alustavat tutkimustulokset pervasiivisten oppimispelien vaikutuksista oppimistuloksiin ovat selkeästi lupaavia, mutta lisätutkimuksia vaaditaan suurempien johtopäätösten vetämiseksi. Opettajien kannalta pelien hyödyntämistä opetuksessa vaikeuttaa tällä hetkellä erityisesti nykyisen pelitarjonnan laadun epätasaisuus [Järvilehto, 2014]. Pervasiivisissa oppimispeleissä tämä vaikeus korostuu, koska tarjonta on epätasaisen lisäksi edelleen myös hyvin vähäistä.

Mainitsin aiemmin ideaalitilanteen olevan, että oppimispelit kannustaisivat myös pervasiiviseen oppimiseen, eli että oppilaat pelaisivat oppimispelejä omaaloitteisesti myös vapaa-ajallaan. Yhtenä suurena ongelmana tässä luonnollisesti on, että pelaamiseen käytettävissä olevasta rajallisesta ajasta kilpailevat oppimispelien lisäksi täysin viihteelliset pelit. Tällä hetkellä, kun viihdepelien markkinointiin ja pelisuunnitteluun käytetään keskimäärin huomattavasti suurempia summia kuin oppimispeleihin, on vaikea nähdä merkittävää muutosta vapaa-ajan pelaamisen suhteen tapahtuvan ainakaan lähitulevaisuudessa. Ehkä Nuclear Mayhemin tapaiset jatkuvasti käynnissä olevat pervasiiviset oppimispelit voisivat toimia esimerkkeinä siitä, kuinka hauskalla tavalla oppiminen siirtyy

sujuvasti myös vapaa-ajalle – vaikka toki kannustin pelaamiseen tässäkin tapauksessa tulee koulusta, peliin osallistumisen ollessa edellytys kurssin läpäisyyn.

Kuten toin esiin, pelkkä oppimistuloksiin keskittyminen ei välttämättä ole riittävä määre pelien hyötyjen arviointiin. Yksi tutkielmani tavoitteista olikin selvittää, voisivatko pervasiiviset pelit tuoda uudenlaista yhteisöllisyyden kulttuuria ja parempaa oppimista kouluihin. Yhteisöllisyyden kokemista on varmasti jopa oppimistuloksia vaikeampaa mitata kvantitatiivisesti, mutta useassa tutkimuksessa [mm. Facer *et al.*, 2004] haastatellut lapset kuitenkin osoittivat innostusta erityisesti ryhmissä pelaamista ja oppimista kohtaan, mistä voi vetää ehkä ainakin varovaisen myönteisiä johtopäätöksiä vaikutuksesta yhteisöllisyyteen.

Hyvää oppimista ei mitenkään voi määrittää yksiselitteisesti, mutta De Corte [2010] on luetellut joitakin hyvän oppimisen ominaispiirteitä, joista vallitsee tutkijoiden joukossa kohtalainen yksimielisyys. Näitä ovat oppimisen konstruktiivisuus, kumulatiivisuus, itseohjautuvuus, tavoitesuuntautuneisuus, tilannesidonnaisuus ja yhteistoiminnallisuus. Pervasiivisilla oppimispeleillä tapahtuva oppiminen on vahvasti konstruktiivista, eli oppilaiden omakohtaista ja aktiivista ymmärtämisen ja merkitysten rakentamista. Kumulatiivisuus riippuu paljon pelistä, mutta ainakin kaikissa käsitellyissä peleissä on kumulatiivisia piirteitä: uuden oppiminen perustuu aiemmin opittuihin tietorakenteisiin. Hiekkalaatikkopeleissä kumulatiivisuuden aste toki riippuu pitkälti opettajan viitseliäisyydestä. Myöskään itseohjautuvuuden tai tavoitesuuntautuneisuuden osalta ei voida vetää kaikkia pervasiivisia pelejä koskevia yleistyksiä. Tilannesidonnaisuuden sen sijaan voi sanoa olevan jopa yksi näiden pelien erityispiirteistä: oppiminen liitetään usein elävään tilanteeseen ja tietoa luodaan ja käsitellään sen aidossa kontekstissa. Yhden määritelmän [Sahlberg ja Leppilampi, 1994] mukaan tilannesidonnainen oppiminen on mielen, kehon, ympäristön ja tilanteen prosessien yhdistelmä, ja tämän voinee olettaa toteutuvan keskimäärin sitä paremmin, mitä useampi pelin taikapiirin rajoista rikotaan. Oppimisen yhteistoiminnallisuutta korostetaan usein konstruktivismin yhteydessä, sillä tietojen prosessointi sosiaalisesti saattaa luoda uusia yhteisiä merkityksiä. Useimmat nykyiset pervasiiviset oppimispelit sisältävät yhteistoiminnallisen elementin.

Lähtökohtaisesti pervasiiviset oppimispelit täyttävät siis De Corten hyvän oppimisen kuuden ominaispiirteen listasta ainakin kaksi. Muiden piirteiden täyttyminen on pelisuunnittelijoiden, opettajien ja joissain tapauksissa ehkä vähän oppilaidenkin osaamisen ja tahdon varassa.

Yksi lupaavimmista tutkimustuloksista on jo mainittu pervasiivisten oppimispelien usein aiheuttama flow-tila. Vaikka pelien vaikutuksista oppimistuloksiin on toistaiseksi melko vähän näyttöä, on flow-tilan ja paremman oppimisen

välillä todettu olevan selkeä yhteys [mm. Woszczynski *et al.*, 2002]. Jälleen pelisuunnittelijoiden vastuulle jää, että flow-tilasta aiheutuva intensiivinen keskittyminen ja motivaatio kohdistetaan pelin tavoitteiden lisäksi myös oppimistavoitteiden saavuttamiseksi.

Olenkin nyt useasti tuonut esiin pelisuunnittelijoiden vastuun pervasiivisten pelien vaikutuksessa oppimistulosten paranemiseen. Mutta ehkä yhtenä syynä nykyisen oppimispelitarjonnan laadun epätasaisuuteen onkin, että harvat pelisuunnittelijat ovat lopulta mitään kasvatusalan ammattilaisia. Koska yhteisöllisen oppimisen teoreettinen tietämys on tärkeässä roolissa pedagogisesti toimivien pervasiivisten pelien luomisessa, esitän näin lopuksi ajatuksen monitieteellisemmästä pelisuunnittelusta. Olisiko nyt aika saada myös kasvatustieteilijöiden ja humanistien ääni paremmin kuuluviin pelialalla? Kokonaisvaltaisen ja kokemuksellisen oppimisen ottaminen pelisuunnittelun pohjaksi vaatii erityisesti kehityspsykologista osaamista. Oppilaiden itsetuntemuksen kehittämiseen, oman arvomaailman pohtimiseen tai sosiaalisten taitojen parantamiseen tähtäävät pelit eivät välttämättä suoraan takaa parempia oppimistuloksia, mutta tuskin humanistisemmasta lähestymistavasta koulutukseen ainakaan haittaa olisi nykypäivänä.

Jokainen pedagoginen teoria on tietyllä tapaa oman aikansa tuote, ja uusia teorioita syntyy maailman muuttuessa, mistä hyvänä esimerkkinä design-suuntautunut pedagogiikka. Tämä antaa myös opettajille enemmän vapauksia kokeilla uusia opetustapoja: perinteisiä oppiainerajoja voidaan rikkoa ja opiskelijoille voidaan tarjota aktiivisemman toimijan rooli. Konnektivismi korostaa verkostojen ja kytkentöjen merkitystä, joten se on loistava lähestymistapa pervasiivisten oppimispelien rakentamiselle. Kun fyysisistä, ajallisista ja välttämättä sosiaalisistakaan rajoista ei enää tarvitse välittää entiseen tapaan, tarjoaa tämä oppilaille mahdollisuuden luoda laajoja verkostoja, joissa yhteiset mielenkiinnon kohteet kytkevät heidät muihin ihmisiin, mikä edesauttaa tehokkaasti uuden oppimista. Kun pelilliset elementit yhdistyvät pervasiivisuuteen, säilyy oppiminen myös hauskana. Yhtenä koulun, ja osin jälleen myös pelisuunnittelijoiden, tärkeänä tehtävänä korostuu tällöin mediakriittisyyden opettaminen.

Teknologia on nykyisin yhä pervasiivisempaa, eikä koulu voi mitenkään sulkeutua kehityksen ulkopuolelle. Teknologia itsessään tosin on pelkkä väline, joten kauhistelemisen sijaan kannattaisikin pyrkiä etsimään ja toteuttamaan sen tarjoamia mahdollisuuksia paremmalle oppimiselle ja ihmisenä kasvamiselle. Pelien opetuskäyttöä kohtaan esitettyä kritiikkiä ei toki tule sivuuttaa: on totta, etteivät opetuksen pelillistäminen ja digitalisointi voi olla mitään itseisarvoja, eikä hyväksi todettuja opetuskäytänteitä tietenkään saa niiden varjolla romuttaa.

Selkeä trendi, jossa oppijat itse ovat yhä useammin oppimisen keskiössä, on nähtävissä. Koulutuksen ja oppimisen paradigmamuutos on käynnissä, ja kukaan tuskin osaa täysin ennustaa, miltä koululuokat tulevat näyttämään tulevina vuosikymmeninä. Pervasiiviset oppimipelit ovat uutena tulokkaana haastamassa nykyisiä ajattelutapoja – miten ne siinä onnistuvat, jää nähtäväksi.

Viiteluettelo

[Admiraal *et al.*, 2011] Wilfried Admiraal, Jantina Huizenga, Sanne Akkerman, and Geert ten Dam, The concept of flow in collaborative game-based learning. *Computers in Human Behavior* **27**, 3 (2011), 1185-1194.

[Avouris and Yiannoutsou, 2012] Nikolaos M. Avouris and Nikoleta Yiannoutsou, A review of mobile location-based games for learning across physical and virtual spaces. *Journal of Universal Computer Science* **18**, 15 (2012), 2120-2142.

[Bateson and Martin, 2013] Patrick Bateson and Paul Martin, *Play, playfulness, creativity and innovation*. Cambridge University Press, 2013.

[Benford *et al.*, 2004] Steve Benford, Duncan Rowland, Martin Flintham, Richard Hull, Jo Reid, Jo Morrison, Keri Facer, and Ben Clayton, Savannah: Designing a location-based game simulating lion behaviour. *International conference on advances in computer entertainment technology*, (2004).

[Bell, 2010] Frances Bell, Connectivism: Its place in theory-informed research and innovation in technology-enabled learning. *The International Review of Research in Open and Distributed Learning* **12**, 3 (2010), 98-118.

[Bransford *et al.*, 1999] John D. Bransford, Ann L. Brown, and Rodney R. Cocking, *How people learn: Brain, mind, experience, and school*. National Academy Press, 1999.

[Caillois and Barash, 1961] Roger Caillois and Meyer Barash, *Man, Play and Games*. University of Illinois Press, 1961.

[Daniels, 2005] Harry Daniels, *An introduction to Vygotsky*. Psychology Press, 2005.

[De Corte, 2010] Erik De Corte, Historical developments in the understanding of learning. In Hanna Dumont, David Istance, and Francisco Benavides (eds.), *The nature of learning. Using research to inspire practice*. OECD Publishing, 2010, 35-67.

[Deterding *et al.*, 2011] Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart Nacke, From game design elements to gamefulness: defining gamification. *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, (2011), 9-15.

[Ejsing-Duun *et al.*, 2013] Stine Ejsing-Duun, Thorkild Hanghøj, and Helle Skovbjerg Karoff, Cheating and creativity in pervasive games in learning

contexts. *7th European Conference on Games Based Learning, Porto, Portugal*, (2013).

[Entertainment Software Association, 2015] Essential facts about the computer and video game industry: 2015 sales, demographic and usage. *Press Release, Entertainment Software Association*, (2015).

[Erhel and Jamet, 2013] Séverine Erhel and Éric Jamet, Digital game-based learning: Impact of instructions and feedback on motivation and learning effectiveness. *Computers & Education* **67**, (2013), 156-167.

[Facer *et al.*, 2004] Keri Facer, Richard Joiner, Danae Stanton, Jo Reid, Richard Hull, and David Kirk, Savannah: mobile gaming and learning?. *Journal of Computer Assisted Learning* **20**, 6 (2004), 399-409.

[Habgood and Ainsworth, 2011] MP Jacob Habgood and Shaaron E. Ainsworth, Motivating children to learn effectively: Exploring the value of intrinsic integration in educational games. *The Journal of the Learning Sciences* **20**, 2 (2011), 169-206.

[Hinske *et al.*, 2007] Steve Hinske, Matthias Lampe, Carsten Magerkurth, and Carsten Röcker, Classifying pervasive games: on pervasive computing and mixed reality. *Concepts and Technologies for Pervasive Games: A Reader for Pervasive Gaming Research* **1**, (2007), 11-38.

[Huizenga *et al.*, 2009] Jantina Huizenga, Wilfried Admiraal, Sanne Akkerman, and Geert ten Dam, Mobile game-based learning in secondary education: engagement, motivation and learning in a mobile city game. *Journal of Computer Assisted Learning* **25**, 4 (2009), 332-344.

[Huizinga, 1955] Johan Huizinga, *Homo Ludens: A Study of the Play-elememt in Culture*. Beacon Press, 1955.

[Järvilehto, 2014] Lauri Järvilehto, *Hauskan oppimisen vallankumous*. PS-kustannus, 2014.

[Kittl and Petrovic, 2008] Christian Kittl and Otto Petrovic, Pervasive games for education. *Proceedings of the 2008 Euro American Conference on Telematics and Information Systems*, (2008).

[Malaby, 2007] Thomas M. Malaby, Beyond play: A new approach to games. *Games and Culture* **2**, 2 (2007), 95-113.

[Montola, 2005] Markus Montola, Exploring the edge of the magic circle: Defining pervasive games. *Proceedings of DAC* **1966**, (2005).

[Niemi *et al.*, 2005] Jenny Niemi, Susanna Sawano, and Annika Waern, Involving non-players in pervasive games. *Proceedings of the 4th Decennial Conference on Critical Computing: Between Sense and Sensibility. Aarhus, Denmark*, (2005).

[Oppenheimer, 2007] Todd Oppenheimer, *The flickering mind: Saving education from the false promise of technology*. New York: Random House, 2007.

[Pløhn, 2014] Trygve Pløhn, Pervasive learning – Using games to tear down the classroom walls. *Electronic Journal of e-Learning* **12**, 3 (2014).

[Postman, 1995] Neil Postman, *The End of Education*. New York: Alfred A. Knopf, 1995.

[Postman, 1985] Neil Postman, *Amusing Ourselves to Death: Public Discourse in the Age of Television*. New York: Viking, 1985.

[Sahlberg ja Leppilampi, 1994] Pasi Sahlberg ja Asko Leppilampi, *Yksinään vai yhteisvoimin? Yhdessäoppimisen mahdollisuuksia etsimässä*. Helsingin yliopisto, Vantaan täydennyskoulutuslaitos, 1994.

[Schwabe and Göth, 2005] Gerhard Schwabe and Christoph Göth, Mobile learning with a mobile game: design and motivational effects. *Journal of Computer Assisted Learning* **21**, 3 (2005), 204-216.

[Stenros *et al.*, 2012] Jaakko Stenros, Annika Waern, and Markus Montola, Studying the elusive experience in pervasive games. *Simulation & Gaming* **43**, 3 (2012), 339-355.

[Stenros *et al.*, 2007] Jaakko Stenros, Markus Montola, and Frans Mäyrä, Pervasive games in ludic society. *Future Play 2007 Conference Proceedings. Toronto, Algoma University College & University of Ontario Institute of Technology*, (2007), 30-37.

[Suoninen, 2014] Annikka Suoninen, Lasten mediabarometri 2013. 0-8-vuotiaiden mediankäyttö ja sen muutokset vuodesta 2010. *Nuorisotutkimusverkosto/Nuorisotutkimusseura. Verkkojulkaisuja* **75**, (2014).

[Thomas, 2006] Siobhán Thomas, Pervasive learning games: Explorations of hybrid educational gamescapes. *Simulation & Gaming* **37**, 1 (2006), 41-55.

[Vartiainen *et al.*, 2012] Henriikka Vartiainen, Anu Liljeström, and Jorma Enkenberg, Design-oriented pedagogy for technology-enhanced learning to cross over the borders between formal and informal environments. *Journal of Universal Computer Science* **18**, 15 (2012), 2097-2119.

[Viita ja Alkio, 2014] Aija Viita ja Riku Alkio, Pelilautana koko kaupunki. Teoksessa Leena Krokfors, Marjaana Kangas, ja Kaisa Kopisto (toim.), *Oppiminen pelissä: Pelit, pelillisyys ja leikillisyys opetuksessa*. Osuuskunta Vastapaino, 2014, 220-232.

[Waern *et al.*, 2009] Annika Waern, Markus Montola, and Jaakko Stenros, The three-sixty illusion: designing for immersion in pervasive games. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, (2009), 1549-1558.

[Woszczynski *et al.*, 2002] Amy B. Woszczynski, Philip L. Roth, and Albert H. Segars, Exploring the theoretical foundations of playfulness in computer interactions. *Computers in Human Behavior* **18**, 4 (2002), 369-388.

[Ängeslevä, 2014] Sonja Ängeslevä, Tosielämän minecraftaaminen. Teoksessa Leena Krokfors, Marjaana Kangas, ja Kaisa Kopisto (toim.), *Oppiminen pelissä: Pelit, pelillisyys ja leikillisyys opetuksessa*. Osuuskunta Vastapaino, 2014, 118-132.

Käyttäjähyväksynnän malleista

Merita Lemmetty

Tiivistelmä.

Erilaisiin käyttötarkoituksiin on nykyisin tarjolla lukemattomia teknologisia sovelluksia, joista kaikki eivät kuitenkaan saavuta käyttäjien suosiota. Tutkijoita on kiinnostanut jo yli neljännesvuosisadan ajan, mikä vaikuttaa siihen, ottavatko käyttäjät uuden teknologian käyttöönsä vai eivät. Teknologian hyväksymisprosessia, eli käyttäjähyväksyntää, selittämään on kehitetty erilaisia malleja. Alun perin tutkimukset keskittyivät työympäristöihin, sillä 1980-luvulla teknologia ei ollut osa arkielämää. Yritysten päätöksentekijät olivat ja ovat yhä edelleen kiinnostuneita teknologian hyväksymisestä, sillä investointi teknologiaan on hintavaa ja siinä on omat riskinsä. Teknologian hyväksymismalleista on hyötyä yritysten päätöstentekijöiden lisäksi myös teknologian suunnittelijoille, kehittäjille ja testaajille.

Tässä tutkielmassa kerron käyttäjähyväksynnästä aikaisemman kirjallisuuden pohjalta. Käyttäjähyväksyntämalleista esittelen TAM-, TAM2- ja TAM3-mallit sekä UTAUT- ja UTAUT2-mallit. Lisäksi selvitän käyttäjähyväksyntä mallien saamaa kritiikkiä ja pohdin käyttäjähyväksynnän tulevaisuuden näkymiä.

Avainsanat ja -sanonnat: TAM, UTAUT, käyttäjähyväksyntä, kirjallisuuskatsaus.

1 Johdanto

Kaikki teknologiset innovaatiot eivät menesty käyttäjien keskuudessa. Investoiminen teknologiaan voi parantaa tuottavuutta, kun taas epäonnistuneet järjestelmät voivat aiheuttaa rahallisia menetyksiä tai tyytymättömyyttä käyttäjien joukossa [Venkatesh 2000]. Käyttöönottoprosessiin liittyy olennaisesti uuden järjestelmän hyväksyminen, jota on tutkittu laajasti 1980-luvulta lähtien. Teknologian hyväksymisen selittämiseen on kehitetty erilaisia malleja, joista tunnetuin on Fred Davisin [1985] kehittämä TAM-malli. Alkuperäinen malli on kehitetty työorganisaatioita varten, sillä 1980-luvulla teknologian käyttö ei ollut osa vapaa-aikaa. Vuosien saatossa alkuperäisestä mallista on kehitetty useita erilaisia johdannaisia uusiin käyttökonteksteihin.

Tässä tutkielmassa käytän termiä käyttäjähyväksyntä prosessista, joka edeltää uuden teknologisen sovelluksen käyttöönottoa. Esittelen teknologian hyväksymismalleista TAM-mallin syntymisen ja kehityksen nykypäivään asti sekä alkuperäisen TAM-mallin johdannaisista TAM2- ja TAM3-mallit sekä UTAUT- ja UTAUT2-mallit. Tutkielmassa edetään kronologisessa järjestyksessä.

Lopullisesti teknologian hyväksyminen ilmenee sen tosiasiallisena käyttönä. Tämä tutkielma keskittyy kuitenkin tosiasiallista käyttöä edeltäviin käyttöönottoprosessiin vaiheisiin ja tosiasiallinen käyttö on rajattu työn ulkopuolelle. Kaikki mallit ovat tilastollisia malleja, eikä niillä pystytä tarkasti ennustamaan käyttäjän käyttäytymistä. Mallien kehittämisen ja johdannaisien luomisella on ollut tarkoituksena parantaa niiden selitysvoimaa.

Tutkielmassa kerron myös, mitä kritiikkiä mallit ovat saaneet, ja sivuan lyhyesti aloja, joissa erilaisia hyväksymismalleja hyödynnetään, sekä tarkastelen, miltä teknologian hyväksymismallien ja yleisesti käyttäjähyväksynnän tulevaisuudennäkymät näyttävät.

Tämän tutkielman rakenne on seuraava: luvussa 2 keskitytään TAM-mallin syntymiseen ja esitellään mallin johdannaiset ja luvussa 3 käsitellään UTAUT-malleja. Neljännessä luvussa keskitytään käyttäjähyväksyntämallien kritiikkiin, jonka jälkeen luvussa 5 pohditaan mallien tulevaisuuden näkymiä. Luku 6 on yhteenveto.

2 TAM-mallit

2.1 TAM-mallin edeltäjät: TRA- ja TPB-mallit

Teknologisten järjestelmien yleistyessä työorganisaatioissa syntyi tarve ymmärtää, miksi järjestelmät hyväksytään tai torjutaan. Ilmiötä pyrittiin ensiksi selittämään psykologian avulla. *Perustellun toiminnan teoria* eli TRA-malli (Theory of Reasoned Action) [Ajzen and Fishbein 1980] on TAM-mallin edeltäjä. TRA-mallissa tarkkailun kohteena on käyttäjien käyttäytymisaikomus, eikä niinkään heidän asenteensa tuotetta kohtaan. TRA-mallin teorian mukaan parhaiten käyttäjän suhtautumista voidaan ennustaa hänen käyttäytymisaikomuksensa perusteella, johon käyttäjän asenne vaikuttaa suoraan.

TRA-mallin erinäiset rajoitteet synnyttivät *suunnitellun toiminnan teorian,* eli TPB-mallin (Theory of Perceived Behavioral Control) [Ajzen 1985]. TRA-malliin verrattuna TPB-mallissa on huomioitu käyttäjän tietoinen päätös käyttäytyä tietyllä tavoin, eikä vain pelkkä aikomus, niin kuin TRA-mallissa. Tässäkin mallissa on omat rajoitteensa, kuten se, että se toimii vain silloin, kun osa käyttäytymisestä ei ole tahdonalaista. TPB-malli ei huomioi tiedostamattomia motiiveja.

Tutkimuksissa huomattiin, että TRA- ja TPB- mallien sovittaminen erilaisten teknologisten sovellusten käyttöönoton arviointi tuotti huonoja tuloksia. TRA- ja TPB-mallit olivatkin tarkoitettu vain selittämään ihmisen käyttäytymistä. Syntyi tarve tarkemmalle teorialle, joka keskittyisi juuri teknologisten sovellusten käyttöönottoon vaikuttaviin tekijöihin työorganisaatioissa. Tähän tarkoituksen Davis alkoi kehittää TAM-mallia. [Marangunić and Granić 2015.]

2.2 TAM-malli

Davis tähtäsi TAM-mallia kehittäessään siihen, että malli auttaisi hyväksymisprosessin ymmärtämisessä, mikä tuottaisi uusia näkökulmia järjestelmien suunnitteluun ja toteutukseen. Davisin tavoite oli myös kehittää malli, joka tuottaisi teoriaa käyttäjähyväksyntätestaamisesta, mikä taas auttaisi kehittäjiä arviointi- ja toteutusprosesseissa. [Davis 1985.]

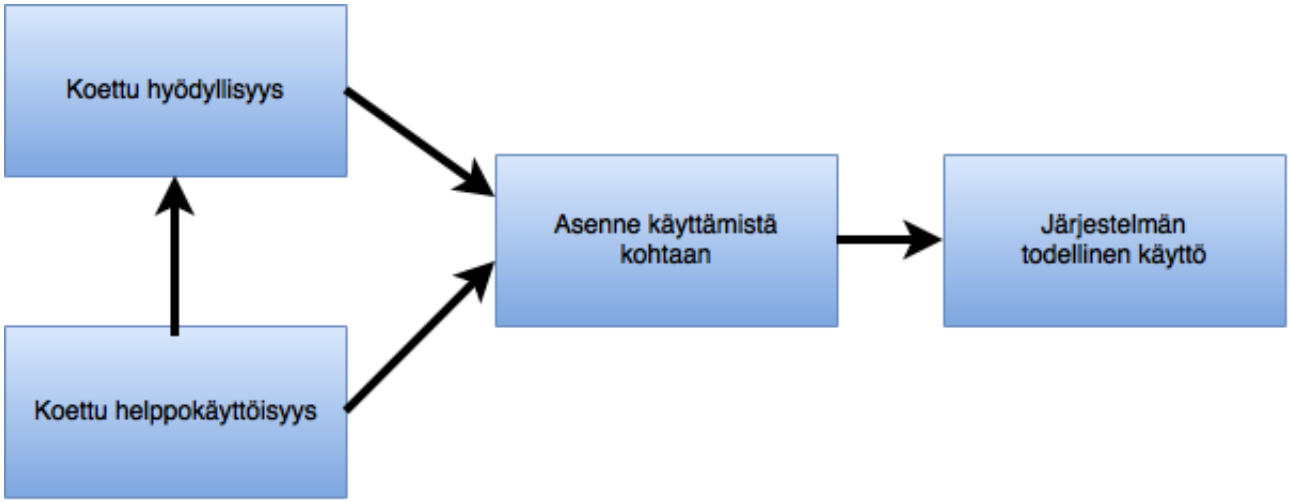
Davis tarkasteli seuraavaa kolmea asiaa: Mitkä ovat ne tekijät (*motivational variables*), jotka selittävät käyttöönottoa järjestelmän ominaisuuksien perusteella? Missä suhteessa nämä tekijät ovat toisiinsa, järjestelmän ominaisuuksiin ja käyttäjän käyttäytymiseen? Miten käyttäjän motivaatiota voidaan mitata, jotta voidaan arvioida käyttäjien käyttäytymisen yhtäläisyyksiä uuden järjestelmän käyttöönotossa? [Davis 1985.]

Davis hyödynsi TRA- ja TPB-mallien taustalla olevia teorioita pyrkiessään vastaamaan mainittuihin tutkimuskysymyksiin. Davisin mukaan edeltävissä malleissa oli päteviä piirteitä, jotka onnistuvat selittämään ja ennustamaan ihmisen käyttäytymistä. Parantaakseen mallien täsmällisyyttä teknologisessa kontekstissa hän ei huomioinut subjektiivista normia, eli muiden henkilöiden vaikutusta käyttäjän mielipiteisiin, käyttäjän todellisessa käyttäytymisessä. Davis keskittyi siis ainoastaan käyttäjän asenteeseen teknologiaa kohtaan.

Davis hyödynsi konseptuaalista mallia hahmottamaan käyttäjän ajatusprosessia, joka on edeltänyt varsinaista käyttöä. Mallin mukaan todellinen järjestelmän käyttö on seurausta käyttäjän motivaatiosta käyttää järjestelmää. Motivaatioon taas vaikuttavat järjestelmän ominaisuudet ja kyvyt (Ks. kuva 1).

Kuva 1. Käyttäjähyväksynnän konseptuaalinen malli [Davis 1985]

Oleellinen tekijä TAM-mallin syntymiseen oli kahden muuttujan eriyttäminen: *koettu hyödyllisyys* (perceived usefulness) ja *koettu helppokäyttöisyys* (perceived ease of use). Davis yhdisti näiden muuttujien vaikutuksen käyttäjän asenteeseen, jotka yhdessä muodostivat alkuperäisen TAM-mallin (Ks. kuva 2). Davisin tekemät muutokset tekivät mallista pätevän ennustamaan käyttäjän asennetta teknologisessa kontekstissa ja ensimmäisissä tutkimuksissa malli selitti 36 % käytöstä. [Davis 1985.]



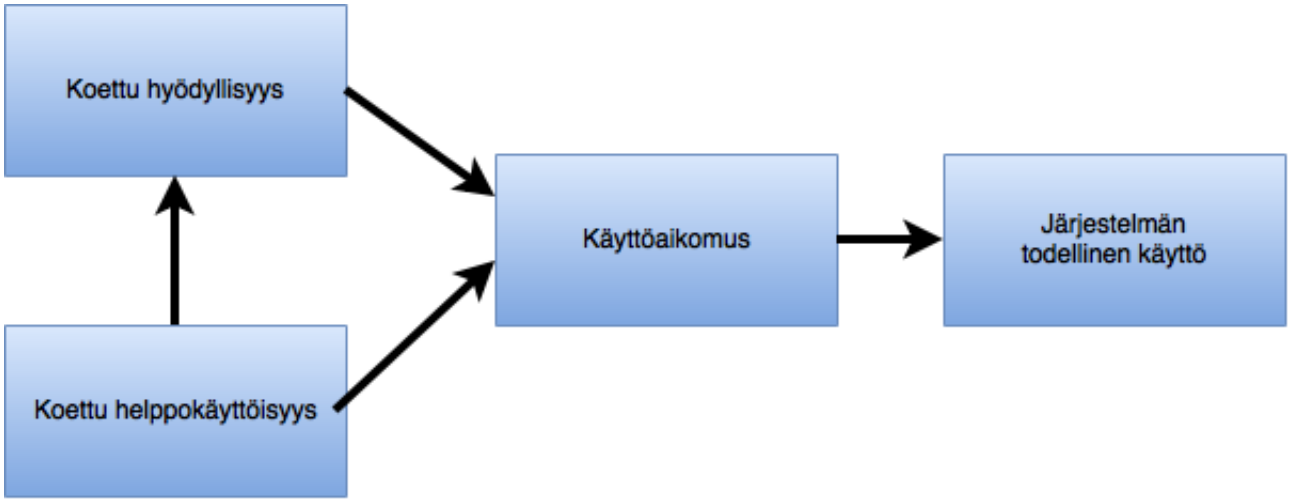
Kuva 2. TAM-malli [Davis 1985]

Seuraavaksi kerron, mitä edellä mainituilla muuttujilla tarkoitetaan tarkemmin. Koettu hyödyllisyys kuvaa, kuinka suureksi käyttäjä kokee järjestelmän käytön antaman hyödyn työskentelyssään. Koettu helppokäyttöisyys kuvaa, kuinka vaivattomaksi käyttäjä kokee järjestelmän käytön. Koettu helppokäyttöisyys perustuu siis käyttäjän arvioon järjestelmän käytöstä aiheutuvaan vaivaan. *Järjestelmän ulkoisilla ominaisuuksilla* (system design characteristics) löydettiin olevan suora vaikutus näihin kahteen muuttujaan. Sekä koettua hyödyllisyyttä että koettua helppokäyttöisyyttä mitattiin kyselylomakkeella, jossa käyttäjän piti arvioida kymmenen erilaista väittämää asteikolla 1-7 sen mukaan, oliko hän väittämän kanssa samaa vai eri mieltä. Esimerkkinä kyselylomakkeen väittämästä on: ”Koen sähköpostin käytön vaivalloiseksi.” [Davis 1985.]

Davis oletti käyttäjän asenteen järjestelmää kohtaa olevan päätekijä siinä, käyttääkö käyttäjä järjestelmää vai ei. Asenteella oletetaankin TAM-mallissa olevan suora vaikutus järjestelmän käyttöön. Käyttäjän asenteeseen taas vaikuttivat koettu hyödyllisyys ja koettu helppokäyttöisyys. Nämä tekijät ovat toisiinsa suhteessa niin, että koetulla helppokäyttöisyydellä on vaikutus koettuun hyötyyn. Toisin sanoen mitä helpompi järjestelmä on käyttää, sen hyödyllisempi sen on. [Davis 1985.] Myöhemmin Davis osoitti, että koetun hyödyllisyyden merkitys nousee suuremmaksi kuin helppokäyttöisyys, eikä helppokäyttöisyys lisää järjestelmän käyttöä, jos käyttäjä ei koe järjestelmää hyödylliseksi [Davis 1989]. Asenteen määrää mitattiin kyselylomakkeen avulla, jossa oli eri väittämiä. Yksi väittämistä oli esimerkiksi: ”Kaikki tekijät huomioiden sähköpostin käyttöni on…”. Käyttäjän piti arvioida väittämä 7-asteisen asteikon avulla. Asteikon toisen pään vaihtoehto on positiivinen ja toinen negatiivinen. Asteikon keskimmäinen aste kuvaa neutraalia suhtautumista. [Davis 1985.]

2.3 Laajennettu TAM-malli

Myöhemmin Davis [1989] huomasi, että koettu hyödyllisyys ja koettu helppokäyttöisyys eivät vaikutakaan suoraan käyttäjän asenteeseen. Seurauksena tästä hän teki ehdotuksen *laajennettuun TAM-malliin* (parsimonious TAM), jossa asenteen vaikutus on poistettu. Myöhemmissä tutkimuksissa korvaavaksi muuttujaksi määriteltiin *käyttöaikomus* (intention to use) (Ks. kuva 3). Koetun hyödyllisyyden todettiin suoraan vaikuttavan käyttöaikomukseen ja selittävän käyttöaikomuksesta yli puolet. Muutosta perusteltiin sillä, että käyttäjällä voi olla vahva aikomus käyttää järjestelmää, jonka hän on kokenut hyödylliseksi ilman asenteen muodostamista. Käyttöaikomusta mitattiin samalla tapaa kuin asennetta, eli kyselylomakkeella, jossa väittämiä arvioitiin 7-asteisella asteikolla. [Davis et al. 1989.]



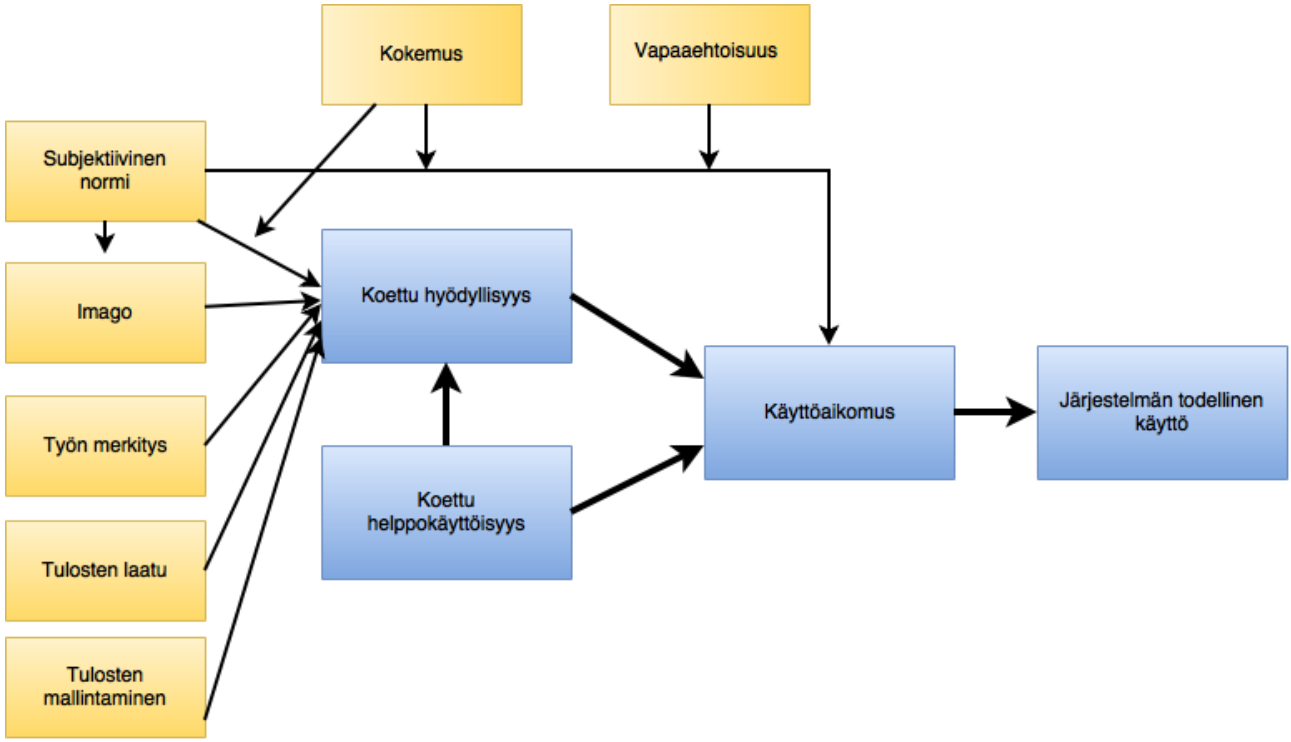
Kuva 3. Laajennettu TAM-malli [Davis et al. 1989]

TAM-mallin muuttujien muutoksen seurauksena koetun hyödyllisyyden suora vaikutus järjestelmän todelliseen käyttöön voitiin selittää paremmin. Muutos aiheutti myös sen, että alettiin pohtimaan uudelleen järjestelmän *ulkoisia muuttujia* (external variables), joilla on vaikutus käyttäjän käsitykseen järjestelmän helppokäyttöisyydestä tai hyödyllisyydestä. Vaikuttavaksi muuttujiksi osoittautuivat jo aiemmin todetun järjestelmän ulkoisten ominaisuuksien lisäksi käyttäjän harjoittelu, käyttäjän osallistuminen suunnitteluun ja käyttöönottoprosessin luonne. [Venkatesh and Davis 1996.]

2.4 TAM2-malli ja koettu hyödyllisyys

Tutkimukset vahvistivat oletusta siitä, että koettu hyödyllisyys on tärkein selittävä muuttuja käyttöaikomuksen arvioinnissa [Davis 1989; Davis et al. 1989]. Venkatesh ja Davis [2000] ehdottivat laajennetun mallin nimeämistä TAM2-malliksi. Malliin sisällytettiin muuttujat, joilla huomattiin olevan vaikutusta koettuun hyödyllisyyteen (Ks. kuva 4). Muuttujat jaettiin kahteen kategoriaan sen perusteella, vaikuttaako muuttuja sosiaaliseen prosessiin vai kognitiiviseen prosessiin. Sosiaaliseen prosessiin vaikuttavilla muuttujilla

tarkoitetaan muiden henkilöiden vaikutusta käyttäjän mielipiteisiin. Näitä ovat TAM2-mallissa subjektiivinen normi ja imago. Kognitiiviseen prosessin muuttujia käytetään kuvaamaan käyttäjän arviota siitä, miten järjestelmä pystyy suoriutumaan hänelle asetetuista työtehtävistä. Muuttujia ovat työn merkitys, tulosten laatu ja mallintaminen sekä koettu helppokäyttöisyys. [Venkatesh and Davis 2000.]



Kuva 4. TAM2-malli [Venkatesh and Davis 2000]

Myös kokemuksella todettiin olevan vaikutusta subjektiiviseen normiin. Kokemuksen kasvaessa subjektiivisen normin vaikutus pieneni, eivätkä käyttäjät muodostaneet mielipidettään järjestelmän hyödyllisyydestä subjektiivisen normin perusteella. Koetun hyödyllisyyden arviointi jatkui entisellään, jos käyttäjät kokivat järjestelmän käytön vaikuttavan heidän imagoonsa.

Vapaaehtoisuudella todettiin olevan suuri merkitys subjektiiviseen normiin. Vaikutuksen merkitys korostui järjestelmissä, joiden käyttö on käyttäjälle pakollista. Tästä johtuen myös vapaaehtoisuus lisättiin yhdeksi muuttujaksi TAM2-malliin. Uudet muuttujat lisäsivät mallin tulosten luotettavuutta ja käyttöaikomukseen vaikuttavia tekijöitä pystyttiin selittämään entistä paremmin. TAM2-malli selittää 34–52 %. käyttöaikomuksesta. [Venkatesh and Davis 2000]

Seuraavaksi kuvaan TAM2-malliin lisättyjen muuttujien merkitykset tarkemmin [Venkatesh and Davis 2000]:

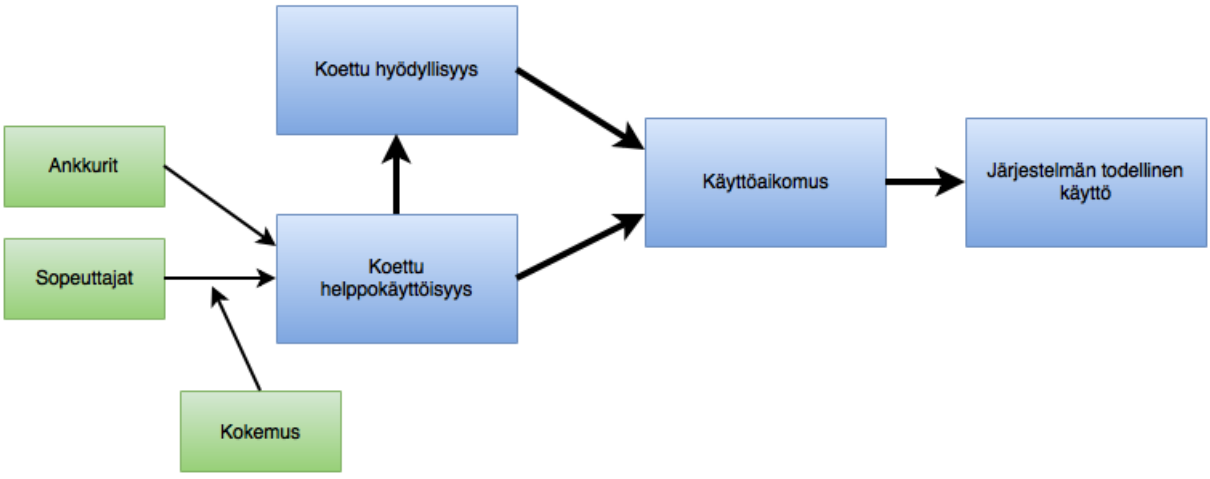
- subjektiivinen normi: muiden henkilöiden vaikutus siihen, tekeekö käyttäjä päätöksen käyttää vai olla käyttämättä järjestelmää

- imago: järjestelmän käytön vaikutus muiden mielikuvaan käyttäjästä
- työn merkitys: hyöty, jonka käyttäjä kokee saavansa järjestelmän käytöstä
- tulosten laatu: arvio siitä kuinka hyvin järjestelmä pärjäsi vaadituissa tehtävissä
- tulosten mallintaminen: järjestelmän käytöstä seuraavien tulosten aistittavuus
- vapaaehtoisuus: onko järjestelmän käyttö käyttäjälle vapaaehtoista vai onko se määrätty hänelle
- kokemus: mielikuvat aikaisemmista käyttökokemuksista.

2.5 Koettu helppokäyttöisyys

Yksi TAM-malliin laajennuksista syntyi Venkateshin [2000] kiinnostuessa määrittelemään, mitkä ovat ne ratkaisevat tekijät, jotka vaikuttavat koettuun helppokäyttöisyyteen. Tätä edeltävät tutkimukset olivat keskittyneet määrittelemään lähinnä koettuun hyödyllisyyteen vaikuttavia tekijöitä. Venkateshin mielestä koettuun helppokäyttöisyyteen vaikuttavien tekijöiden ja sen taustan ymmärtäminen takaisi uusia näkökulmia käyttäjähyväksyntään.

Venkatesh määritteli kaksi pääryhmää, joilla on vaikutusta koettuun helppokäyttöisyyteen: *ankkurit* (anchors) ja *sopeuttajat* (adjustments) (Ks. kuva 5). Ankkurit kuvaavat yleisiä uskomuksia tietokoneista ja niiden käytöstä, joita käyttäjällä on ennen todellista käyttöä. Sopeuttajilla tarkoitetaan käyttäjän käsitystä, johon tarkasteltavan järjestelmän käyttö on vaikuttanut. Suora käyttökokemus kohdejärjestelmän kanssa saa käyttäjän sopeuttamaan käsityksensä järjestelmän helppokäyttöisyydestä. Toisin sanoen käyttäjien oletetaan muodostavan ankkurit aikaisemman käytön perusteella ja heidän oletetaan sopeuttavansa käsityksensä järjestelmän vuorovaikutuksen myötä. [Venkatesh 2000.]



Kuva 5. Ankkureiden ja sopeuttajien suhde helppokäyttöisyyteen [Venkatesh 2000]

Venkatesh määritteli ankkurit ja sopeuttajat sekä niihin vaikuttavia tekijöitä. Hänen mukaansa ankkurit ovat yksilöiviä ja tilannekohtaisia. Niiden kehittymiseen vaikuttavat

kontrolli, luontainen motivaatio ja tunteet. Kontrolli voidaan jakaa vielä käsityksiin sisäisestä ja ulkoisesta kontrollista. Sisäinen kontrolli ilmenee siinä, miten käyttäjä luottaa omiin kykyihinsä järjestelmän käytössä ja ulkoinen kontrolli siinä, miten järjestelmään on toteutettu käytön mahdollistavat edellytykset. Luontainen motivaatio havainnollistuu käyttäjän leikkimielisyyden kautta. Leikkimielinen suhtautumien järjestelmää kohtaan saa käyttäjän suhtautumaan järjestelmään tutkistelevammin ja uteliaammin. Leikkimielisten käyttäjien on todettu arvioivan järjestelmän käytön helpommaksi kuin ne, jotka ovat vähemmän leikkimielisiä. Tunteista tietokonepelolla on suurin vaikutus siihen, millaiseksi käyttäjän ennakko-oletukset, eli ankkurit, järjestelmän helppokäyttöisyydestä muotoutuvat.

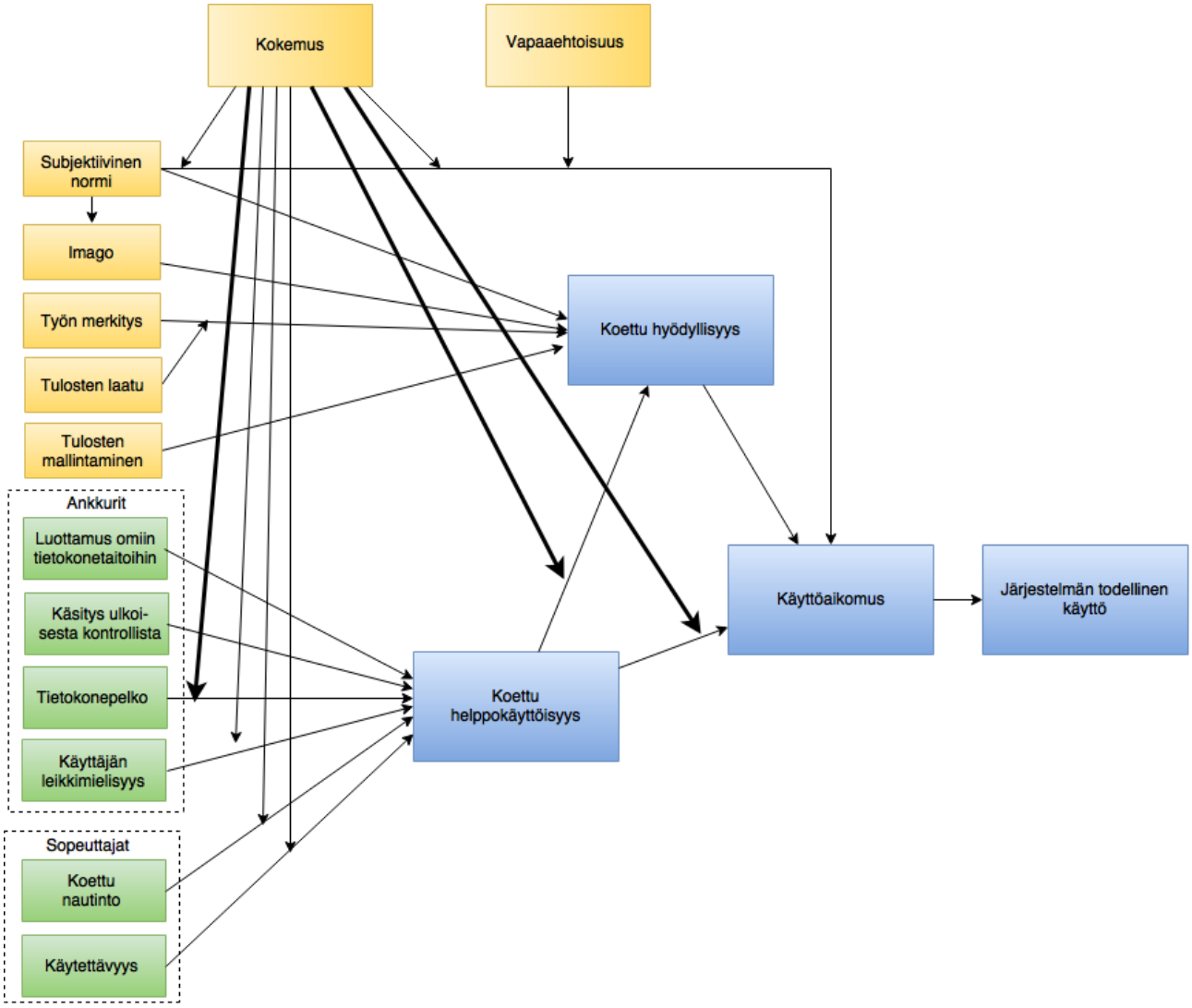
Edellä esittelin ankkureihin vaikuttavia tekijöitä. Seuraavaksi käsittelen, mitkä tekijät vaikuttavat sopeuttajien muodostumiseen. Kuten edellä totesin, sopeuttajat muotoutuvat käyttökokemuksen kertyessä. Venkateshin mielestä sopeuttajiin vaikuttavat käyttäjän mielipide järjestelmän käytettävyydestä ja käsitys ulkoisesta kontrollista. Ulkoinen kontrolli ilmenee kunkin järjestelmän ympäristöstä ja koetusta nautinnosta, joka syntyy käyttäjän käyttäessä järjestelmää. [Venkatesh 2000.]

Tutkimukset osoittivat Venkateshin mallin olevan luotettava ja pätevä. Malli selittää koettua helppokäyttöisyyttä jopa 60 %, joka on kaksi kertaa enemmän, mihin edeltävät mallit ovat pystyneet. [Venkatesh 2000.]

2.6 TAM3-malli

Edellä esittelin Venkateshin ja Davisin [2000] tutkimia koettuun hyödyllisyyteen vaikuttavia tekijöitä ja Venkateshin [2000] tutkimia koettuun helppokäyttöisyyteen vaikuttavia tekijöitä. Nämä tutkimukset ovat tehty erillään toisistaan, eikä malleissa ole huomioitu mahdollisia ristikkäisvaikutuksia, eli miten koettuun hyödyllisyyteen vaikuttavat tekijät vaikuttavat koettuun helppokäyttöisyyteen tai miten koettuun helppokäyttöisyyteen vaikuttavat tekijät vaikuttavat koettuun hyödyllisyyteen.

Venkatesh ja Bala [2008] ryhtyivät tutkimaan aikaisempien tutkimustulosten mahdollisia ristikkäisvaikutuksia. Tutkimuksissa ilmeni, että niitä ei ole. He loivat TAM3-mallin yhdistämällä edellisissä tutkimuksessa löydetyt tekijät ja lisäsivät malliin kolme uutta päämuuttujien välistä suhdetta, joihin kokemuksella löydettiin olevan vaikutus (Ks. kuva 6). Uudet suhteet ovat koetun helppokäyttöisyyden ja koetun hyödyllisyyden välinen suhde, tietokonepelon ja koetun helppokäyttöisyyden välinen suhde sekä koetun helppokäyttöisyyden ja käytön aikomuksen välinen suhde. Kuvassa 6 uusia suhteita korostamaan on käytetty paksumpaa viivaa. TAM3-malli onnistui selittämään käyttöaikomuksesta 40–53 %.



Kuva 6. TAM3-malli [Venkatesh and Bala 2008]

Kokemuksen vaikutus koetun helppokäyttöisyyden ja koetun hyödyllisyyden välillä ilmenee siten, että käyttökokemuksen karttuessa käyttäjän on helpompi arvioida, kuinka helppo tai vaikea järjestelmä on käyttää. Arvio helppokäyttöisyydestä vahvistaa käyttäjän arviota hyödyllisyydestä.

Tietokonepelon ja helppokäyttöisyyden suhteeseen kokemuksella on heikentävä vaikutus, eli mitä enemmän käyttäjällä on kokemusta, sitä vähemmän käyttäjä kokee tietokonepelkoa ja arvioi järjestelmän helppokäyttöisemmäksi. Kokemuksen lisääntyessä uskomukset muuttuvat yleisistä uskomuksista enemmän järjestelmäkohtaisiksi.

Kokemuksella on heikentävä vaikutus myös koetun helppokäyttöisyyden ja käyttöaikomuksen suhteeseen. Käyttäjän sopeutuessa järjestelmän käyttöön koetun helppokäyttöisyyden rooli pienenee, eli käyttäjät eivät pohjusta enää käyttöaikomustaan koetun helppokäyttöisyyden määrään. [Venkatesh and Bala 2008.]

Venkatesh ja Bala [2008] pyrkivät TAM3-mallia luodessaan mahdollisimman kokonaisvaltaiseen malliin, jonka muuttujat selittäisivät ne tekijät, joilla on vaikutus käyttäjähyväksyntään yksilöllisellä tasolla. Mallin yhteydessä he selvittivät toimenpiteitä, jotka tukisivat TAM3-mallin teoriaa ja tehostaisivat teknologisten järjestelmien käyttöönottoprosessia työntekijöiden joukossa. Löydetyt toimenpiteet jaettiin käyttöönottoa edeltäviin ja sitä tukeviin toimenpiteisiin. Käyttöönottoa edeltäviksi toimenpiteiksi Venkatesh ja Bala luokittelivat järjestelmän suunnitteluprosessin, käyttäjien osallistumisen, johtoportaan tuen ja kannustimien asettamisen. Käyttöönottoa tukevia toimenpiteitä heidän mukaansa ovat koulutus, työorganisaation tarjoama tuki ja vertaistuki.

3 UTAUT-mallit

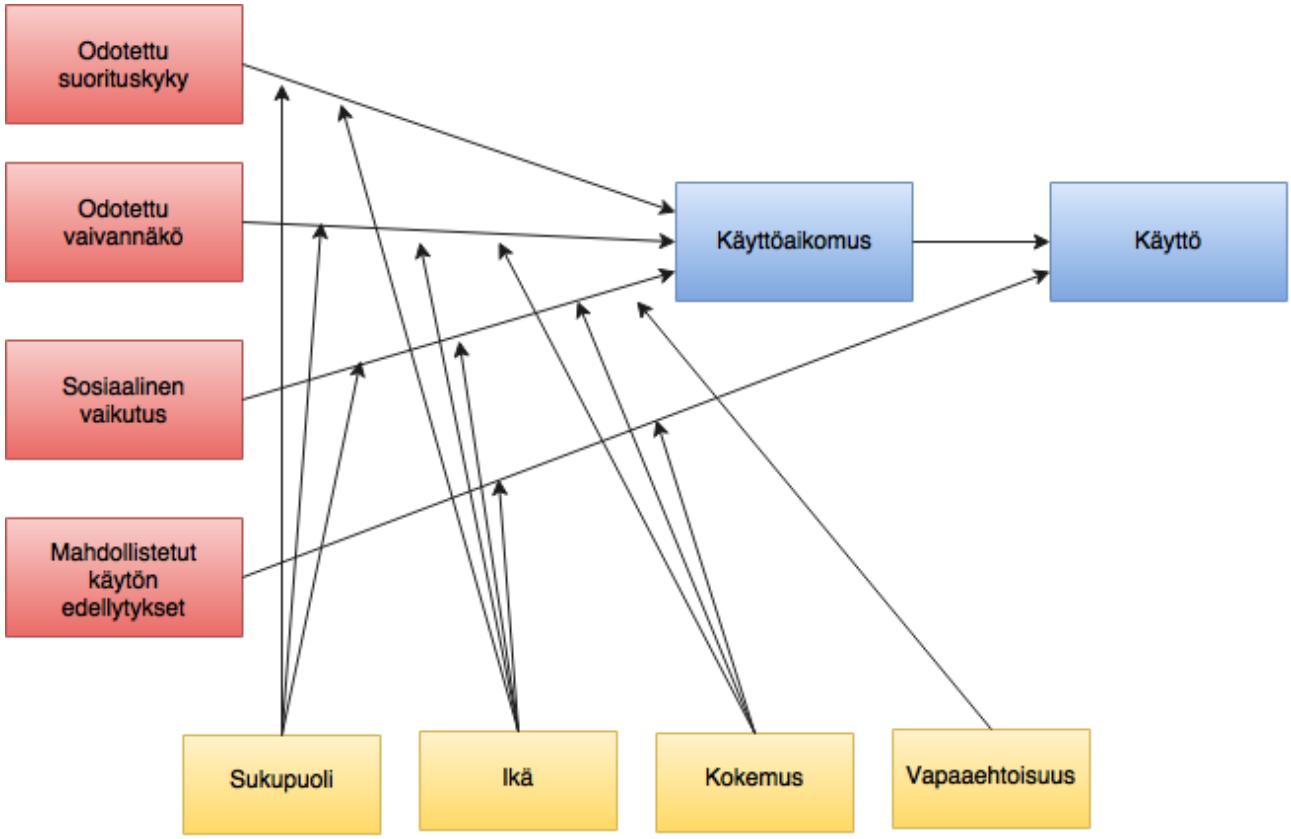
3.1 UTAUT-malli

Venkatesh ja muut [2003] vertasivat kahdeksaa eri käyttäjähyväksyntää arvioivaa mallia ja niiden laajennuksia. Näihin kahdeksaan malliin sisältyivät tässä tutkielmassa aiemmin esitellyt TRA- sekä TAM-mallit. Vertailujen perusteella luotiin malli, johon oli yhdistetty kaikissa kahdeksassa toistuneet elementit. Kyseinen malli nimettiin *UTAUT-malliksi* (Unified Theory of Acceptance and Use of Technology) ja se selittää käyttöaikomuksesta jopa 70 %. UTAUT-malli antaa näkemyksen siitä, kuinka käyttöaikomuksen ja käytöksen päätekijät ovat kehittyneet. Mallin tavoitteena oli tarjota yritysten johtoportaalle hyödyllinen väline arvioimaan uuden järjestelmän käyttöönoton onnistumista työntekijöiden keskuudessa ja löytämään ne tekijät, jotka vaikuttavat olennaisesti teknologian hyväksymiseen. [Venkatesh et al. 2003.]

Tutkimuksissa nousi esille neljä tekijää, joilla oli selvä suora vaikutus käyttöaikomukseen ja käyttöön. Nämä muuttujat ovat odotettu suorituskyky, odotettu vaivannäkö, sosiaalinen vaikutus ja mahdollistetut käytön edellytykset (Ks. kuva 7). Odotettu suorituskyky kuvaa sitä, kuinka paljon käyttäjä uskoo järjestelmän käytön hyödyttävän hänen työsuoritustaan. Odotettu vaivannäkö kuvaa sitä vaivaa, joka järjestelmän käytöstä aiheutuu. Sosiaalinen vaikutus kuvaa sitä, miten käyttäjä kokee hänelle tärkeiden ihmisten näkevän hänen kykynsä käyttää järjestelmää. Tämä muuttuja muistuttaa aiemmin TAM2-mallissa esitettyä subjektiivisen normin käsitettä. Mahdollistetut käytön edellytykset kuvaavat sitä, kuinka paljon käyttäjä kokee, että järjestelmään on toteutettu rakenteita käytön tukemiseksi.

Odotettu suorituskyky, odotettu vaivannäkö ja sosiaalinen vaikutus kohdistuvat käyttöaikomukseen. Käyttöaikomuksella yhdessä mahdollistettujen käytön edellytyksien kanssa on suora vaikutus todelliseen käyttöön.

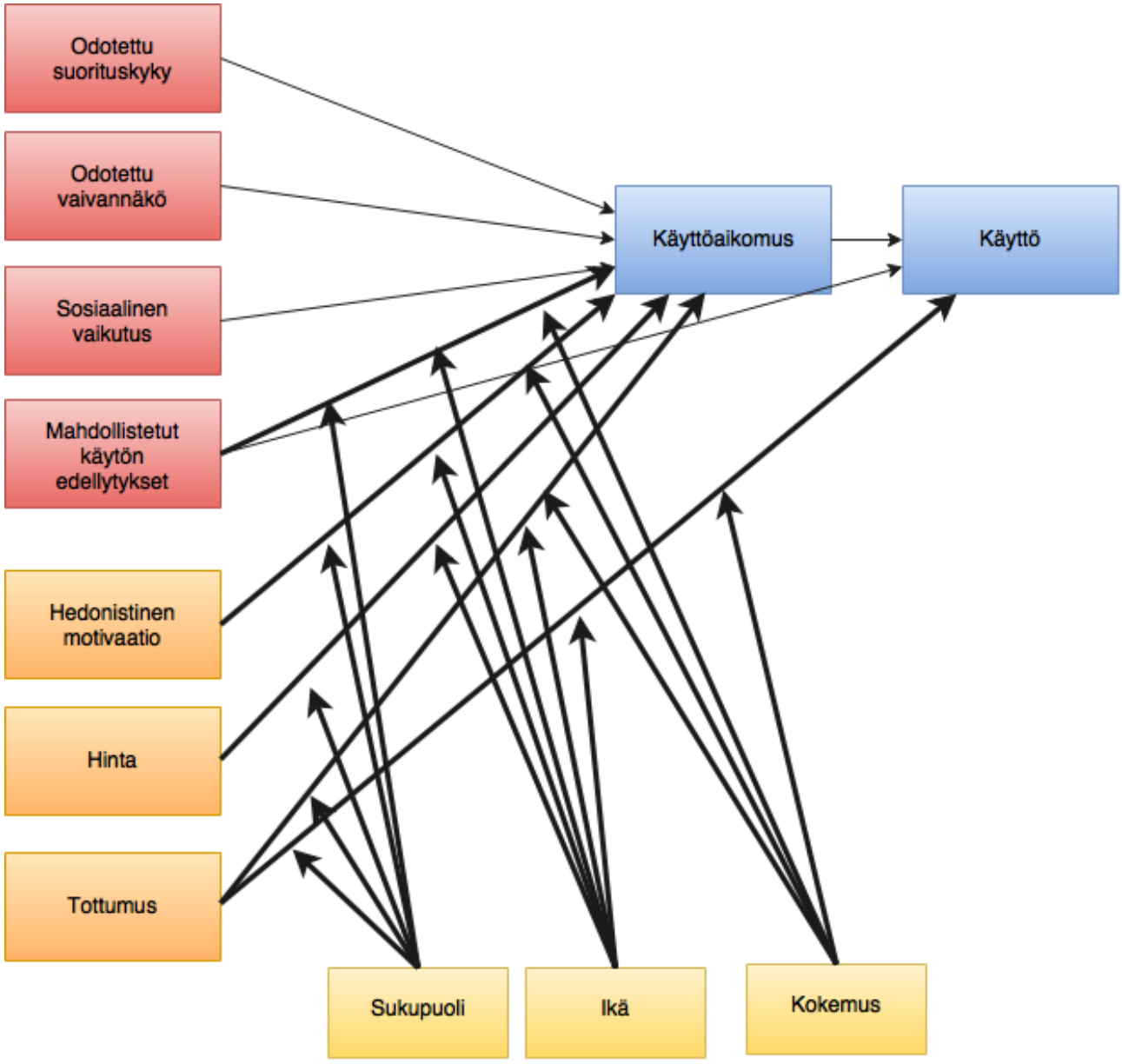
Neljän päämuuttujan lisäksi malliin lisättiin tekijät, joilla on vaikutus päämuuttujien suuruuteen. Nämä lisämuuttujat ovat sukupuoli, ikä, vapaaehtoisuus ja kokemus.



Kuva 7. UTAUT-malli [Venkatesh et al. 2003]

3.2 UTAUT2-malli

Venkatesh ja muut [2012] laajensivat UTAUT-mallia soveltumaan kuluttajakontekstiin, ja malli nimettiin UTAUT2-malliksi. Malli selittää käyttöaikomuksesta 56–74 %. Malliin lisättiin seuraavat kolme päämuuttuja: hedonistinen motivaatio, hinta ja tottumus (Ks. kuva 8). Kuten UTAUT-mallissa, myös UTAUT2-mallin päämuuttujiin vaikuttavat lisämuuttujat, joita ovat sukupuoli, ikä ja kokemus. Koska UTAUT2-malli on suunniteltu kuluttajakontekstiin, oletetaan käyttämisen olevan vapaaehtoista. Tästä syystä vapaaehtoisuus poistettiin kokonaan lisämuuttujista. Erona edeltävään UTAUT-malliin käytön mahdollistavien tekijöiden ja käyttöaikomuksen väliin muodostettiin myös suhde, kun aiemmin käytön mahdollistavilla tekijöillä oli vaikutus ainoastaan käyttöön.



Kuva 8. UTAUT2-malli [Venkatesh et al. 2012]

Kuvaan 7 on korostettu uudet suhteet paksummalla viivalla ja kuvan selkeyttämiseksi vanhat UTAUT-mallin suhteet on jätetty piirtämättä. Edelleen odotettuun suorituskykyyn vaikuttavia lisämuuttujia ovat ikä ja sukupuoli, odotettuun vaivannäköön vaikuttavat ikä, sukupuoli ja kokemus, sosiaaliseen vaikutukseen vaikuttavat ikä, sukupuoli ja kokemus sekä mahdollistettuihin käytön edellytyksiin vaikuttavat ikä ja kokemus.

Seuraavaksi kerron, mitä uusilla päämuuttujilla tarkoitetaan ja perustelen, miksi ne on lisätty UTAUT2-malliin. Hedonistinen motivaatio kuvaa mielihyvää, jota järjestelmän käyttö käyttäjälle antaa. Hedonistinen motivaatio on todettu olevan yksi päätekijä, joka vaikuttaa teknologian hyväksymiseen ja käyttöön.

Toisin kun työelämän kontekstissa, käyttäjät ovat kuluttajina vastuussa kuluista. Kuluilla on vaikutus käyttäjien päätöksentekoon. UTAUT-malli keskittyi ainoastaan aikaan ja

vaivannäköön, joten hinta-muuttujan lisäys täydentää erilaisten resurssien huomioonottamista. Hinta suhteutetaan teknologian käytöstä aiheutuvaan hyötyyn. Jos saatu hyöty on suurempi kuin rahallinen kustannus, on hinnalla positiivien vaikutus käyttöaikomukseen.

Tottumus lisättiin malliin täydentämään tahallista tai suunniteltua käyttäytymistä. Tottumuksella on todettu olevan suora vaikutus teknologian käyttöön. Se voi vaikuttaa käyttöaikomuksen ja käytön suhteeseen kasvattavasti tai vähentävästi. Tottumuksella kuvataan käyttäjän automaatista toimintaa, joka on tapahtunut oppimisen seurauksena. Tottumuksella on kaksi erilaista ulottuvuutta. Se voi kuvata käyttäjän ensisijaista toimintatapaa tai se voi kuvata käyttäjän automaattista toimintatapaa. UTAUT2-mallissa tottumuksella kuvataan aiempien kokemusten perusteella muokkautunutta toimintatapaa.

UTAUT2-mallissa on säilytetty samat päämuuttujat kuin UTAUT-mallissa, mutta niiden määritelmä on sovellettu sopimaan paremmin kuluttajakontekstiin. Odotettu suorituskyky kuvaa sitä hyötyä, jota järjestelmän käyttö takaa käyttäjälle. Odotettu vaivannäkö kuvaa teknologian helppokäyttöisyyden määrää. Sosiaalinen vaikutus kuvaa käyttäjälle tärkeiden ihmisten mielipidettä siitä, pitäisikö käyttäjän käyttää teknologiaa. Mahdollistetut käytön edellytykset kuvaavat neuvoja ja tukea, jota on saatavilla, jotta käyttö on mahdollista. [Venkatesh et al. 2012.]

Lisätyistä päämuuttujista hedonistisella motivaatiolla ja hinnalla on vaikutus käyttöaikomukseen. Tottumus vaikuttaa käyttöaikomuksen lisäksi myös suoraan käyttöön.

4 Kritiikki

TAM-malli on viitatuin tutkimuksen kohde käyttäjähyväksynnän alalta [Marangunić and Granić 2015], mutta se on saanut osakseen myös kritiikkiä. Mallia on syytetty muun muassa siitä, että se on sivuttanut käyttäjähyväksynnän todelliset ongelmat ja mallin kehityksessä on käytetty vain helppoja ja nopeita tutkimusmenetelmiä [Chuttur 2009]. On kritisoitu, että mallista puuttuu muuttujia, jotka kuvaisivat inhimillisten ja sosiaalisten muutosten prosessia [Legris et al. 2003]. Mallia on tosin laajennettu kyseisen kritiikin jälkeen ja uusilla muuttujilla on pyritty huomioimaan muun muassa juuri näitä asioita (esim. TAM3- tai UTAUT-malli). TAM-mallissa on myös paljon ristiriitaisia näkemyksiä riippuen teoreettisista oletamuksista ja käytännöllisestä tehokkuudesta. Tämä kielii siitä, että mallin taustaoletukset kaipaavat perusteellisempaa tutkimusta. [Chuttur 2009.]

Eri tutkimuksissa on vaihdellen käytetty teknologian hyväksymisen arviointiin avainmuuttujana TAM-mallin mukaista järjestelmän käyttöä tai TRA-mallin mukaista käyttöaikomusta. Järjestelmän käyttö voidaan jakaa todelliseen, raportoituun ja arvioituun käyttöön. Eri jakoperusteet aiheuttavat eroja mittaustapoihin tutkimusten välillä ja lisäksi rajanveto näiden välille on haastavaa. [Wu and Du 2012.] Legris ja muut [2003] korostivat, että TAM-mallien tutkimuksessa mitattu käyttö perustuu käyttäjän tekemään arvioon käytön

määrästä, joten he väittävät TAM-mallin mittavan enemmän vaihtelua raportoidun käytön välillä, mikä selvästikään ei ole tavoiteltu lopputulos.

TAM-mallit on alun perin suunniteltu työorganisaatioiden käyttöön, mutta eri mallien luotettavuuden testaukseen on käytetty paljon opiskelijoita. Onkin pohdittu, ovatko eri mallien selitysasteet täysin luotettavia todellisessa käyttöympäristössä [Legris et al. 2003].

Wu ja Du [2012] osoittivat mallien, joiden käytön ennustaminen perustuu käyttöaikomukseen eikä käyttöön, ennustavan järjestelmän hyväksymistä huonommin. Heidän mielestään käyttöaikomusta ei pidä käyttää korvamaan käyttö-muuttujaa, vaan luotettavimman tuloksen takaamiseksi tutkijoiden tulisi jokaisessa tutkimuksessa arvioida erikseen todellisen käytön ja arvioidun käytön vaikusta selvittääkseen todelliset suhteet järjestelmän käytön ja sen edeltäjien välillä.

Käyttäjähyväksyntämalleihin liittyen on keskustelu myös siitä, kummalla on suurempi vaikutus käyttöaikomukseen, koetulla hyödyllisyydellä vai koetulla helppokäyttöisyydellä. Davisin [1989] mukaan koetun hyödyllisyyden merkitys nousee suuremmaksi kuin helppokäyttöisyys, kun taas Venkatesh ja muut [2003] toteavat, ettei helppokäyttöisyydellä ole niin suurta roolia järjestelmän käytön myöhemmässä vaiheessa. Venkateshin ja Balan [2008] mukaan taas käyttäjät muodostavat käsityksensä hyödyllisyydestä nimenomaan helppokäyttöisyyden perusteella.

5 Käyttäjähyväksynnän tulevaisuus

Teknologian hyväksymistä on tukittu jo yli neljännesvuosisadan ajan, mutta Marangunić ja Granić [2015] ennustavat, että teknologian kasvava tarve niin työelämässä kuin vapaa-ajalla pitää yllä kiinnostusta tutkia käyttäjähyväksyntää vielä useiden vuosien ajan. Erilaisia teknologian hyväksymismalleja kehitetään jatkuvasti ja pyritään parantamaan mallien ennusteiden paikkansapitävyyttä. Teknologia-alan jatkuvan kehityksen myötä käyttäjien monimuotoisuus ja uusien teknologiasuuntauksien määrä kasvaa. Tämä luo tarvetta käyttäjähyväksynnän ymmärtämiselle ja edistää potentiaalisten TAM-mallien kehittämistä.

Kun TAM-mallia sovelletaan uuteen kontekstiin, mallin muuttujia kehitetään sopimaan paremmin uuteen ympäristöön. Ongelmaksi nouseekin, missä menee TAM-mallin laajenemisen raja ja onko muuttujia vaihdellessa enää oikeutettua puhua TAM-mallin johdannaisista. Rajanveto termin käytöstä on hankalaa.

Marangunić ja Granić [2015] tekivät kirjallisuuskatsauksen käyttäen 85 eri käyttäjähyväksyntää käsittelevää lähdettä. Katsaus osoitti seuraavat neljä potentiaalista suuntaa, joissa lisätutkimus olisi tarpeen: yksittäisen muuttujan heikentävä vaikutus teknologian hyväksymiseen (esimerkiksi teknologiapelko), uusien muuttujien lisääminen malliin (esimerkiksi muuttujien, jotka selittävät eri kulttuuritaustoista tulevien henkilöiden suhtautumista teknologiaan), todellisen käytön objektiivisesta mittaamisesta (perustuvatko käyttötiedot käyttäjien oman ilmoituksen sijaan esimerkiksi järjestelmän käytöstä tallennettuun lokiin) sekä

vanhuksille suunniteltujen teknologisten sovellusten tutkiminen (esimerkiksi kuinka mobiilisovellukset voivat auttaa muistisairauksissa). Sharp [2009] taas on määritellyt potentiaalisiksi tulevaisuuden tutkimuskohteiksi vapaaehtoisten ja pakollisten ympäristöjen aiheuttamat erot, asenteen rooli käyttäjähyväksynnässä sekä kritiikki-osiossakin esille nostetun koetun helppokäyttöisyyden ja koetun hyödyn ristiriitaiset tulokset siitä kumman vaikutus on vahvempi.

Kun Davisin ensimmäinen teknologian hyväksymistä selittävä TAM-malli oli kehitetty työorganisaatioihin, on ajan saatossa mallia sovellettu uusiin ja tarkempiin käyttökonteksteihin. Luvussa 3 kerroin UTAUT2-mallista, jonka sovellusala on kuluttajakonteksti. Listaa jatkaakseni tutkijoita ovat kiinnostaneet myös esimerkiksi mobiililaitteiden ja -palveluiden käyttöönotto [Laforet and Li 2005; Pagani 2008], verkko-oppimiseen liittyvät aiheet [Roca et al. 2006; Persico et al. 2014] sekä hoivarobottien käyttäjähyväksyntä [Broadbent et al. 2009]. Lista eri sovellusaloista on pitkä ja uusien teknologioiden syntyessä se kasvaa jatkuvasti.

6 Yhteenveto

Käyttäjähyväksyntä on kiinnostanut tutkijoita jo yli neljännesvuosisadan. Tänä aikana teknologia on kokenut murroksen, ja ennen vain työelämässä yleinen teknologia on muuttunut osaksi arkielämäämme. Kilpailu teknologian alalla on kovaa ja siksi on tärkeää ymmärtää, millainen teknologinen sovellus tai järjestelmä saavuttaa käyttäjien suosion. Teknologian hyväksymismalleja onkin käytetty apuna sovelluskehityksessä ja testausprosesseissa.

Ensimmäiset käyttäjähyväksyntää selittävät mallit olivat suunniteltu selittämään teknologian hyväksymistä työympäristössä ja mallien oli tarkoitus auttaa organisaatioiden päätöksentekijöitä. Näistä ensimmäinen on Davisin vuonna 1985 kehittämä TAM-malli. Mallia on tutkittu laajasti ja sille on kehitetty monia erilaisia johdannaisia.

Tutkijat ovat pyrkineet kehittämään malleja, joiden avulla käytön tai käyttöaikomuksen selittäminen olisi tarkempaa. Olennaisena tekijänä mallien kehityksessä on ollut teknologian siirtyminen työympäristöistä osaksi arkipäivää, jolloin käyttö on muuttunut vapaaehtoiseksi. Uusimmat mallit ovat keskittyneet selittämään käyttäjähyväksyntää erilaisissa käyttökonteksteissa, esimerkkinä tästä tässäkin tutkielmassa esittelemäni UTAUT2-malli, joka on kehitetty kuluttajakontekstiin.

Tässä tutkielmassa esittelin TAM-mallin lisäksi TAM2- ja TAM3-mallit sekä UTAUT- ja UTAUT2-mallit. Seuraavaksi kertaan lyhyesti teknologian hyväksymismallien käymän kehityskaaren. Mallien kehityksen yhteydessä on havaittavissa jatkuva kehitys käyttöaikomuksen selitysasteessa. Viimeisenä esitelty UTAUT2-malli selittää käyttöaikomuksesta jopa 74 %. Käyttöaikomusta ja muita muuttujia, kuten koettua hyödyllisyyttä tai koettua helppokäyttöisyyttä, on mitattu kyselylomakkeiden avulla.

Ensimmäisen TAM-mallin keskeisin idea on, että käyttöaikomus vaikuttaa todelliseen käyttöön. Käyttöaikomukseen vaikuttavia tekijöitä ovat taas koettu hyödyllisyys ja koettu

helppokäyttöisyys. Tutkimuksissa on ilmennyt ristiriitaista tietoa siitä, kummalla muuttujista on merkittävämpi rooli. Uusimissakin malleissa on nähtävissä alkuperäisen TAM-mallin ydin, mikä kertoo siitä, että jo ensimmäisen TAM-mallin idea on ollut onnistunut.

TAM-mallin pohjalta kehitettiin TAM2-malli. Oleellinen muutos TAM2-mallissa edelliseen verrattuna on sellaisten tarkempien muuttujien määritteleminen, joilla on vaikutusta koettuun hyödyllisyyteen. TAM2-mallia seurannut TAM3-malliin on lisätty myös koettuun helppokäyttöisyyteen vaikuttavat ankkurit ja sopeuttajat.

UTAUT-malli kehitettiin erilaisella lähestymistavalla. Malli luotiin tutkien sitä edeltävää kahdeksaa teknologian hyväksymismallia, joista alkuperäinen TAM-malli oli yksi. Näistä kahdeksasta mallista yhdistettiin olennaiset ja toistuvat tekijät niin, että käyttöaikomuksen selittäminen on tarkempaa. Seuraavaan UTAUT2-malliin lisättiin muutama päämuuttuja ja poistettiin vapaaehtoisuus-muuttuja, jolloin mallin selitysvoima kuluttajakontekstissa parani.

Teknologia on kehittynyt jatkuvasti, ja tarve käyttäjähyväksynnän tutkimiselle ei näytä hiipuvan. Päinvastoin, tarve tutkimukselle uusissa käyttökonteksteissa on kasvava. Esimerkkeinä tulevaisuuden tutkimuksen kohteista tässä tutkielmassa mainitsin vanhuksille suunnatut mobiilipalvelut, yleisesti mobiililaitteet ja -palvelut, verkko-oppiminen sekä hoivarobotit. Vaikka käyttäjähyväksyntämallien kehittämiselle on kasvava tarve, on pohdittava, missä tulee TAM-mallin laajentamisen rajat vastaan. Tulisi miettiä, kuinka paljon yhteen malliin on järkevää sijoittaa muuttujia selittämään käyttäjien hyväksymisprosessia vai pystyisikö prosessin selittämistä lähestymään kokonaan toisesta, uudesta näkökulmasta.

Lähteet

Ajzen, I. and Fishbein, M. 1980. *Understanding Attitudes and Predicting Social Behavior*. Prentice Hall.

Ajzen, I. 1985. From intentions to actions: a theory of planned behavior. In: Kuhl, J., Beckmann, J. (eds.), *Action Control: From Cognition to Behavior*, 11-39.

Broadbent , E., Stafford, R. and MacDonald B. 2009. Acceptance of Healthcare Robots for the Older Population: Review and Future Directions. *International Journal of Social Robotics* 1, 4, 319-330.

Chuttur, M.Y. 2009. Overview of the technology acceptance model: origins, developments and future directions. *Sprouts: Working Papers on Information Systems* 9, 37, 1–21.

Davis, F.D. 1985. *A Technology Acceptance Model for Empirically Testing New End-User Information Systems: Theory and Results.* Doctoral dissertation. MIT Sloan School of Management.

Davis, F.D. 1989. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly* 13, 3, 319-340.

Davis, F.D., Bagozzi, R.P. and Warshaw, P.R. 1989. User acceptance of computer technology: a comparison of two theoretical models. *Management Science* 35, 8, 982–1003.

Laforet, S. and Li, X. 2005. Consumers' attitudes towards online and mobile banking in China. *International Journal of Bank Marketing* 23, 5,362-380.

Legris, P., Ingham, J. and Collerette, P. 2003. Why do people use information technology? A critical review of the technology acceptance model. *Information & Management* 40, 3, 191-2014.

Marangunić, N. and Granić, A. 2015. Technology acceptance model: a literature review from 1986 to 2013. *Universal Access in the Information Society* 14, 1, 81-95.

Pagani, M. 2008. Determinants of adoption of third generation mobile multimedia services. *Journal of Interactive Marketing* 18, 3, 46-59.

Persico, D., Manca, S. and Pozzi, F. 2014. Adapting the Technology Acceptance Model to evaluate the innovative potential of e-learning systems. *Computers in Human Behavior* 30, 614–622.

Roca, J., Chiu, C. and Martínez, F. 2006. Understanding e-learning continuance intention: An extension of the Technology Acceptance Model. *International Journal of Human-Computer Studies* 64, 8, 683-696.

Sharp, J.H. 2007. Development, extension, and application: a review of the technology acceptance model. *Information Systems Education Journal* 5, 9, 1–11.

Venkatesh, V. 2000. Determinants of perceived ease of use: integrating control, intrinsic motivation, and emotion into the technology acceptance model. *Information Systems Research* 11, 4 , 342–365.

Venkatesh, V. and Bala, H. 2008. Technology Acceptance Model 3 and a Research Agenda on Interventions. *Decision Sciences* 39, 2, 273-315.

Venkatesh, V. and Davis, F.D. 1996. A model of antecedents of perceived ease of use: development and test. *Decision Sciences* 27, 3, 451–481.

Venkatesh, V. and Davis, F.D. 2000. A theoretical extension of the technology acceptance model: four longitudinal field studies. *Management Science* 46, 2, 186–204.

Venkatesh, V., Morris, M. G., Davis, G. B. and Davis, F. D. 2003. User acceptance of information technology: Toward a unified view. *MIS quarterly* 27, 3, 425-478.

Venkatesh, V., Thong, J.Y.L. and Xu, X. 2012. Consumer Acceptance and Use of Information Technology: Extending the Unified Theory of Acceptance and Use of Technology. *MIS Quarterly* 36, 1, 157–178.

Wu, J. and Du, H. 2012. Toward a better understanding of behavioral intention and system usage constructs. *European Journal of Information Systems* 21, 6, 680-698.

Software product line engineering -paradigman testauksen ongelmat

Jarmo Multanen

Tiivistelmä.
Software product line engineering -paradigma on suosittu ohjelmistokehitysratkaisu kehitettäessä useita ohjelmia, jotka jakavat samoja ominaisuuksia. Tässä tutkielmassa pyrin kirjallisuuskatsauksen avulla saamaan yleiskuvan kirjallisuudessa käsitellyistä menetelmistä paradigman testauksessa. Havaintoni on, että paradigman testauksen käytännöissä on vielä paljon tutkittavaa ja parannettavaa.

Avainsanat ja -sanonnat: Software product line engineering, testaus, vaatimusmäärittely, massakustomointi, alusta.

1. Johdanto

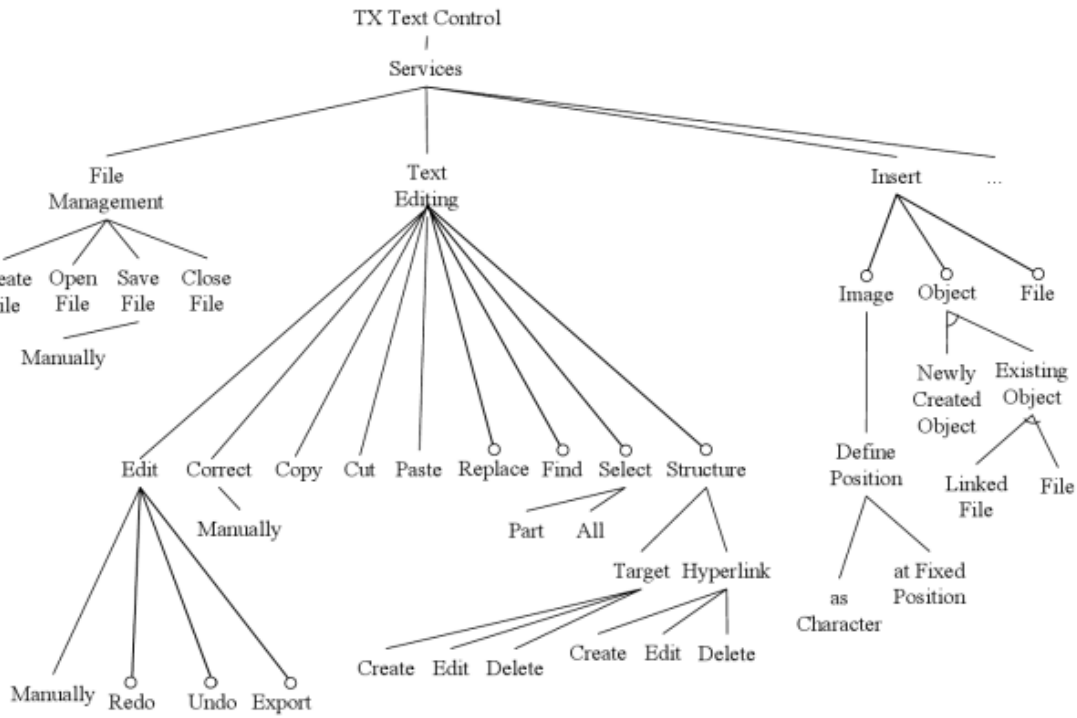
Nykypäivänä ohjelmistoihin kohdistuu enenevässä määrin erilaisia vaatimuksia. Ohjelmistojen tulisi olla käyttäjilleen yksilöllisiä, sillä kaikilla käyttäjillä ei ole samoja vaatimuksia ja käyttötarkoituksia ohjelmistojen suhteen, mutta samaan aikaan niiden tulee olla yritykselle kustannustehokkaita ja nopeita kehittää. Jokaiselle yksittäiselle asiakkaalle oman tuotteen teko ei kuitenkaan ole järkevää, joten on täytynyt kehittää paradigma näiden haasteiden ylittämiseen. Pohl ja muut [2005, 4] käyttävät ohjelmistojen vaatimusten vertauskuvana autojen valmistuksen kehitystä. Liukuhihnan keksimisen ja yleistymisen myötä autojen valmistus nopeutui huomattavasti, mutta samalla tuotteiden yksilöllisyys väheni, sillä autot valmistettiin identtisiksi samoista osista. [Pohl *et al.* 2005, 4].

Kuluttajat kuitenkin tarvitsevat autoja eri käyttötarkoituksiin, kuten kuljetukseen tai perhekäyttöön, joten erilaisten mallien kehittäminen muodostui tärkeäksi tuotannolliseksi seikaksi. Tähän vaatimukseen vastaa *massakustomointi* (mass customization). Se määritellään sellaisten tuotteiden valmistukseksi, jotka vastaavat yksilöidyn asiakkaan tarpeita. [Pohl *et al.* 2005, 4]

Massakustomointi ei kuitenkaan itsessään ole kustannustehokasta, sillä jokaiselle asiakkaalle oman yksilöidyn tuotteen tekeminen alusta asti suuressa mittakaavassa on vaativaa. Tästä syystä yritykset kehittivät erilaisia *alustoja* (platform), joiden artefakteiksi kutsuttuja osia muuttamalla pystyttiin valmistamaan erilaisia tuotteita, jotka kuitenkin jakoivat samoja osia. Näin saatiin aikaan tuot-

teiden yksilöllisyys, kuitenkaan lisäämättä tuotantokustannuksia. Alusta määritellään tässä yhteydessä miksi tahansa pohjateknologiaksi, jonka päälle rakennetaan muita teknologioita tai prosesseja. [Pohl *et al.* 2005, 5-6]

Ohjelmistokehityksen alueella yhtä alustan ja massakustomoinnin yhdistävää paradigmaa kutsutaan nimellä Software product line engineering (lyhenne SPLE tai SPL). Pohl ja muut [2005, 14] määrittelevätkin SPLE:n ohjelmistokehitysparadigmaksi, jossa käytetään alustoja ja massakustomointia. Massakustomoinnin avulla on mahdollista käyttää samoja artefakteja eri ohjelmissa. Tätä joustavuutta kutsutaan *muuttuvuudeksi* (variability) [Pohl *et al.* 2005, 8]. Pohl ja muut [2005, 60] määrittelevät muuttuvuuden osa-alueet SPLE:n kontekstissa *muuttuvuussubjekteiksi* (variability subject) ja *-objekteiksi* (variability object). Jos käytännön esimerkiksi otetaan väri, niin värin käsite itsessään on muuttuvuussubjekti ja eri värit, kuten punainen ja sininen, ovat muuttuvuusobjekteja. Näin ollen muuttuvuussubjekti on abstrakti asia, jonka ilmentymiä ovat muuttuvuusobjektit. *Variaatiopiste* (variation point) puolestaan on muuttuvuussubjektin havainnollistus jossain kontekstissa. Esimerkiksi auton väri on variaatiopiste, jossa auto on konteksti ja väri muuttuvuussubjekti. [Pohl *et al.* 2005, 60-62] SPLE-paradigmaa havainnollistetaan yleisesti *ominaisuusmalleilla* (feature model), jotka antavat selkeän kuvan ohjelmiston ominaisuuksista, niiden yhdistelmistä ja variaatiopisteistä [Machado *et al.* 2014]. Kuvassa 1 on ominaisuusmalli, jossa on käytetty esimerkkinä tekstieditoria [Pohl *et al.* 2005, 297].



Kuva 1. Ominaisuusmalli [Pohl *et al.* 2005, 297]

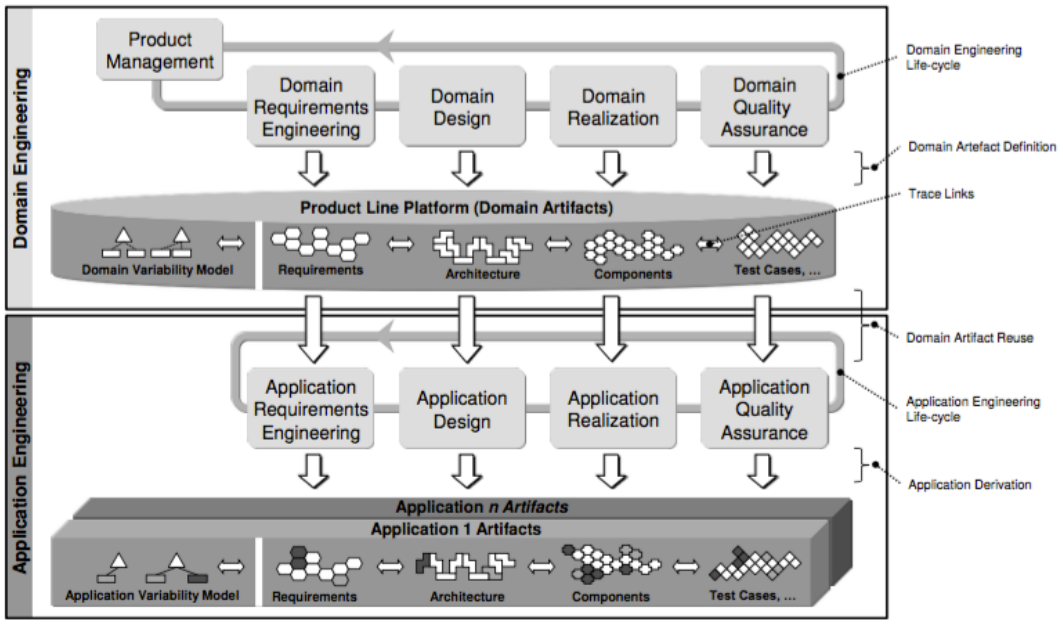
SPLE on ollut yritysten suosiossa vuosia sen tarjoamien hyötyjen takia [Thüm *et al.* 2014]. Se ei ole ainoa tapa lähestyä suuria ohjelmistoprojekteja, mutta se on yksi suosituimmista [Metzger and Pohl 2014]. Pohl ja muut [2005] listaavat SPLE:n hyödyiksi muun muassa kehityskustannusten pienenemisen, laadun parantumisen, kehitysajan lyhenemisen, ylläpidon vaivan vähentymisen ja jatkokehittämisen helppouden.

Paradigman testaus on kuitenkin osoittautunut ongelmalliseksi, ja testaus on jossain määrin vielä kehittymätöntä [Engström and Runeson 2011]. Kirjallisuudessa on esitetty lukuisia testaustapoja, mutta ne eivät aina ole ottaneet kaikkea huomioon tai vastanneet tärkeimpiin kysymyksiin [Lee *et al.* 2012]. Testaustapoja vertaavia tutkimuksia ei ole myöskään juuri tehty [Machado *et al.* 2014].

Tässä tutkielmassa analysoin kirjallisuuskatsauksen avulla SPLE-paradigman testausta, testauksen ongelmia ja kirjallisuudessa ehdotettuja ratkaisuja. Luvussa 2 käsittelen paradigman testauksen yleisiä käytäntöjä ja sitä, miten sen testaus eroaa yhden yksittäisen ohjelmistotuotteen testauksesta. Tämän jälkeen siirryn luvussa 3 käsittelemään kirjallisuudessa esiin tulleita ongelmia testausmetodeissa, jonka jälkeen käsittelen ehdotettuja ratkaisumalleja luvussa 4. Tarkoituksena on tunnistaa kirjallisuudesta paradigman testaukseen liittyviä puutteita ja niihin ehdotettuja ratkaisuja. Luvussa 5 on keskustelu tutkimuksen tulevaisuudesta ja luku 6 sisältää yhteenvedon.

2. SPLE:n testauksen käytännöt

SPLE-paradigman testaus eroaa käytännöiltään yhden ohjelmiston testauksesta. Pohl ja muut [2005, 20-36] jakavat viitekehyksessään SPLE-paradigman kahteen eri kehitysprosessiin: *aluekehitykseen* (domain engineering) ja *sovelluskehitykseen* (application engineering). Aluekehityksessä kehitetään ohjelmistolle alusta ja sovelluskehityksessä kehitetään alustan pohjalta halutut sovellukset. Jako kehitysprosesseihin on havainnollistettu kuvassa 2. Molempiin kehitysalueisiin kuuluu asianmukainen testaus, joka kuvassa 2 on sisällytetty laadunvalvontalaatikkoihin.



Kuva 2. SPLE:n kehitysprosessi [Metzger and Pohl 2014]

Aluekehityksen testauksessa testataan ohjelmiston komponentteja niille asetettuja vaatimuksia vastaan. Näitä ovat muun muassa arkkitehtuuri ja artefaktit. Testauksessa kehitetään myös uudelleenkäytettäviä testiartefakteja, jotta itse sovelluskehityksessä säästettäisiin aikaa. Aluekehityksen testaus eroaa yksittäisen ohjelman kehityksestä siinä, että testausvaiheessa ei vielä ole varsinaisesti olemassa sovellusta, jonka voisi ajaa ja testata, vaan vain yksittäisiä komponentteja voidaan testata. On kuitenkin mahdollista kehittää esimerkkiapplikaatio, jolla voi testata näitä yksittäisiä ohjelman osia. [Pohl *et al.* 2005, 27-28]

Sovelluskehityksen yhteydessä puolestaan testataan itse sovellusta sille asetettuja vaatimuksia vastaan. Suurimmat erot yksittäisen ohjelman kehitykseen ovat lisätestit viallisten kokoonpanojen havaitsemiseksi sekä se, että testauksessa pitää ottaa huomioon sekä uudet ohjelmaa varten kehitetyt osat että vanhat kierrätetyt osat. [Pohl *et al.* 2005, 33-34]

Käytännön esimerkkinä Pohl ja muut [2005, 266-267] käyttävät automaatioohjelman asennusta taloon. Aluekehityksen testaukseen kuuluu yhden oven mekanismien testaus: avaaminen, sulkeminen ja oven tilan tarkistaminen. Sovelluskehityksen testaukseen taas kuuluvat kaikkien asennettujen ovien testaus yhdessä koko automaatiojärjestelmän kanssa. [Pohl *et al.* 2005, 266-267]

Monia erilaisia SPLE-paradigman testausstrategioita on kehitetty vuosien varrella. Machadon ja muiden [2014] mukaan yksi suosituimmista on CIT (combinatorial interaction testing). Strategiaa käytetään siinä vaiheessa ohjelmistoprojektia, kun pyritään määrittelemään, mitkä komponentit pitäisi testata. CIT-strategiassa vähennetään testattavien tapausten määrää sillä periaatteella, että

virhetilanteita aiheuttavat vain muutamien muuttujien vuorovaikutus keskenään ohjelmistoissa. Näin muodostetaan kaikista mahdollisista virheitä aiheuttavien muuttujien yhdistelmistä pareja (pairwise testing), joiden yhdistelmiä testataan. Näin testaus on huomattavasti nopeampaa kuin jos testattaisiin kaikkien muuttujien kaikkia mahdollisia yhdistelmiä. [Perrouin *et al.* 2012]

3. SPLE:n testauksen ongelmat

3.1. Paradigman testausongelmien lähteet

Paradigmaa käyttäen kehitettyjen ohjelmistotuotteiden testaus ei kuitenkaan ole ongelmatonta. Engström ja Runeson [2011] jakavat paradigman testauksen haasteet kolmeen pääluokkaan: 1) miten hallita suuri määrä testejä, 2) miten tasapainottaa työ aluekehityksen ja sovelluskehityksen välillä ja 3) miten hallita muuttuvuutta.

Suuri määrä erilaisia osia tuo mukanaan suuren määrän vaadittavia testejä. Engström ja Runeson [2011] sekä Machado ja muut [2014] mainitsevat, että kaikkien ohjelmiston osien testaus on yleensä käytännössä mahdotonta. Tämä johtuu siitä, että tuotevariaatioiden määrä kasvaa eksponentiaalisesti jokaisen variaatiopisteen kohdalla. Tästä syystä turhaa testausta tulisi vähentää ja testiartefakteja tulisi käyttää uudelleen. [Engström and Runeson 2011]

Tasapainotus kahden eri kehitysprosessin, eli alue- ja sovelluskehityksen, välillä on myös hankalaa, sillä on vaikea arvioida, kuinka monta osaa pitäisi testata ja kuinka monessa eri ohjelmavariantissa. Kehittäjien tulisi myös päättää, missä kehitysvaiheessa mitäkin ohjelmiston osaa testataan, jotta vältyttäisiin turhalta testaukselta. Lisäksi alue- ja sovelluskehitys on voitu jakaa organisaation eri osille, mikä voi hankaloittaa testitapausten arviointia. [Engström and Runeson 2011]

Myös muuttuvuus asettaa haasteita, sillä kehittäjien pitää miettiä, montako eri variaatiopistettä pitäisi testata. Kehittäjien pitäisi myös pitää huolta siitä, ettei lopputuotteeseen pääse ominaisuuksia, joiden ei siellä ole tarkoitus olla. Samat ominaisuudet voidaan saada eri ohjelmissa aikaan eri tavoilla, joten ei myöskään voida olettaa, että jos omaisuus toimii yhdessä ohjelmassa, niin se toimisi myös toisessa. [Engström and Runeson, 2011] *Muuttuvuuden hallinta* (variability management) onkin yksi SPLE-paradigmalle ominainen piirre. Chen and Babar [2011] kutsuvatkin muuttuvuuden hallintaa yhdeksi SPLE:n perustavanlaatuisista aktiviteeteista, joka erottaa sen muista ohjelmistokehityksen lähestymistavoista.

Vaikka muuttuvuuden hallinta on oma osansa kehitysprosessia eikä se liity itsessään testaukseen, voidaan kuitenkin argumentoida, että sillä on hyvin suuri vaikutus paradigman testaukseen. Engström and Runeson [2011] mainitsivat

kolmantena haasteena SPLE-paradigman testaukselle juuri muuttuvuuden sekä Machado ja muut [2014] puhuvat kaikkien mahdollisten testitapausten huomioonoton mahdottomuudesta varsinkin taloudellisesta näkökulmasta. Näin muuttuvuuden hallinnalla on suuri merkitys paradigmalla kehitettävien ohjelmistojen testaukselle, sillä niiden hallinnalla saadaan rajattua muuttuvuutta niin, että testaamisesta tulee järkevää.

3.2. Aukot testauksessa

Testauksesta ja sen ongelmista on kirjoitettu paljon. Machado ja muut [2014] tekivät systemaattisen kirjallisuusarvion aikaisemmista tutkimuksista saadakseen tietoa olemassa olevista keinoista, aukoista tutkimuksessa ja kehittääkseen viitekehyksen uusille tutkimuksille. He valitsivat kaksi näkökulmaa: ominaisuuspohjainen sekä tuotepohjainen näkökulma. Ominaisuuspohjainen näkökulma tarkastelee ohjelmistotuotteiden ominaisuuksien testausta, kun taas tuotepohjainen näkökulma lopullisten tuotteiden testausta. Nämä näkökulmat valittiin, sillä heidän mukaansa iso osa aiemmasta tutkimuksesta on tehty näiden kahden näkökulman mukaan. [Machado *et al.* 2014]

Merkille pantavaa Machado ja muiden [2014] mukaan on se, että vaikka testaukseen käytetyn ajan minimoinnista on kirjoitettu paljon, niin silti ei ole puhuttu kovinkaan paljoa virheiden havaitsemisen parantamisesta. Tämä on yllättävää, sillä yritysten halutessa vähentää testaukseen käytettävää aikaa olisi loogista kehittää testaustapoja, joilla saataisiin tehokkaammin huomattua virheet tuotannossa.

Muita heikkouksia ominaisuuspohjaisesta näkökulmasta katsottuna aikaisemmassa tutkimuksessa on painotus optimointialgoritmeihin, jotka eivät kuitenkaan anna selvää hyötyä. Myöskään vertailevia tutkimuksia eri testaustapojen välillä ei juuri julkaista, eikä mallien toimivuutta varmisteta. [Machado *et al.* 2014]

Tuotepohjaisesta näkökulmasta katsottuna taas ohjelmiston ominaisuuksien testikäytäntöjen ylläpitoa ei käsitellä, vaikka näiden ominaisuuksien määrä isossa projektissa voi kasvaa lukemaan, jota on mahdoton realistisesti testata. Myöskään ohjelmien luonnolliseen evoluutioon, eli kehittymiseen päivitysten myötä, ei juuri puututa. [Machado *et al.* 2014] Näiden asioiden valossa Machado ja muut [2014] tulevat siihen lopputulokseen, että SPLE:n testaus on metodologisesti heikkoa ja lisää empiiristä tutkimusta tarvitaan, jotta voidaan kunnolla selvittää, mikä on paras testaustapa.

Myös Engström ja Runeson [2011] toteavat, että SPLE-paradigman testaus on kehittymätöntä. Testaus ja sen ongelmat tuntuvat olevan keskusteluaihe ja on

olemassa selvä ymmärrys siitä, mitkä asiat ovat ongelmia testauksessa. Kuitenkin suurin osa heidän analysoimistaan tutkimuksista sisältää vain ehdotuksia ja harvassa tutkimuksessa käytetään tai arvioidaan näitä ehdotuksia. He huomauttavat, että paradigman testaamiseen liittyvä empiirinen tutkimus on kallista ja suurmittaista, mikä selittää tutkimuksen puutetta. SPLE-paradigman käytöstä on kokemusta yrityksissä, mutta kuitenkaan tutkimuksia testauksesta ei näytä löytyvän yritysmaailmasta. [Engström and Runeson 2011]

Lee ja muut [2012] puolestaan havaitsivat, että suurin osa heidän arvioimistaan tutkimuksesta keskittyi liian kapeisiin tutkimushaasteisiin. Tästä seurasi se, että tutkimukset esittivät ongelmat yksityiskohtaisesti, mutta eivät käsitelleet niitä koko SPLE -paradigman testausprosessin näkökulmasta [Lee *et al.* 2012]. He jakoivat viitekehyksessään olemassa olevan tutkimuksen arvioinnin kahdeksaan näkökulmaan (testitapausten luonti, testitapausten valinta, puuttuvien muunnelmien testaus, muuttuvuuden sitovuus testauksessa, sovelluskohtainen testaus, työkalutuki ja lähestymistavan mittakaavan muutos). Perusteena jaolle ovat paradigmalle keskeiset operaatiot. He tutkivat, miten hyvin nämä näkökulmat on otettu huomioon olemassa olevassa tutkimuksessa. Kuvassa 3 on nähtävissä tutkimuksen tulos. [Lee *et al.* 2012]

Perspectives	Not	Marginally	Partially	Fully
P1.Test case creation			√	
P2.Test case selection		√		
P3.Test execution for absent variants		√		
P4.Variability binding in testing		√		
P5.Application-specific tests	√			
P6.Tool support		√		
P7.Scalability of the approach	√			
P8.Evidences of the approach		√		

Kuva 3. Kirjallisuusarvioinnin tulos [Lee *et al.* 2012]

Näkökulmat jaettiin neljään luokkaan (ei otettu huomioon, marginaalisesti, osittain sekä kokonaan) riippuen siitä, kuinka hyvin ne on huomioitu. Yhtäkään näkökulmaa ei oltu otettu täysin huomioon. Ainoastaan testitapausten luonti on otettu osittain huomioon tutkimuksissa. Sovelluskohtainen testaus ja mittakaavan muutoksia ei oltu lainkaan huomioitu ja loput näkökulmat on otettu huomioon vain marginaalisesti. [Lee *et al.* 2012] Näkökulmat havainnollistavat hyvin

paradigman tutkimuksen nykytilaa. Varsinkin mittakaavan muutosten huomiotta jättäminen on valitettavaa, kun kyseessä on kuitenkin paradigman yksi perustavanlaatuisista haasteista. On todella vaikea arvioida ratkaisutavan hyödyllisyyttä, jos sen soveltuvuutta erikokoisiin ohjelmistoprojekteihin ei ole otettu tutkimuksissa huomioon.

Loppujen lopuksi on yllättävää, miten puutteellisesti SPLE-paradigman testausta on. Usea kirjallisuusarviointi päätyi samaan lopputulokseen, kuitenkin painottaen eri näkökumia. Se, ettei luotettavia testikeinoja ole kehitetty tai tuloksia verrattu, jättää suuren aukon sekä paradigman testauksen tutkimukseen että käytäntöihin. Myös Metzger ja Pohl [2014] korostavat paradigman nykytilan kartoituksessaan uusittavissa olevan empiirisen tutkimuksen puutetta, ja turhien testien vähentämisen ohella he pitävät tutkimuksen puutetta yhtenä isoimpana haasteena SPLE-paradigmalle testauksen puolella.

4. Ratkaisuja

Edellä mainituille ongelmille on pyritty eri tutkimuksissa keksimään toimivia ratkaisuja. Ratkaisuehdotukset on valittu siksi, että ne vastaavat kirjallisuusarvioinneissa esitettyihin ongelmiin ja kritiikkeihin. Monien ehdotuksien ongelma on kuitenkin se, ettei niitä ole vertailtu toisiin ehdotuksiin tai toistettu.

4.1. FMT -mallinnuskieli

Schürr ja muut [2010] esittävät yleisesti tuotantolinjojen testauksen kuvaamiseen käytettävien CT-mallien (Classification Tree) ja tuotantolinjojen yleiseen kuvailuun käytettävien ominaisuusmallien yhdistämistä FMT-malliksi (Feature Model for Testing). Schürr ja muut [2010] huomasivat tarkastelunsa aikana, että yleisesti SPLE-paradigmassa näitä erikseen käytettäviä malleja yhdistää moni asia ja että ne voitaisiin yhdistää, sillä molemmat mallit saavat toistensa vahvuudet käyttöön sekä suunnittelussa että testauksessa. Schürr ja muiden [2010] mukaan FMT-mallia käyttävällä lähestymistavalla on seuraavat edut verrattuna perinteiseen ominaisuusmallia ja CT-mallia erikseen käyttävällä lähestymistavalla: testiparametrien luontiheuristiikat voidaan helposti mukauttaa uusiin tuotelinjan testattaviin ohjelmiin, algoritmien kehitys kustannustehokkaiden testitapausten laskemiseen ja optimointiin mahdollistuu sekä manuaaliset toiminnot testitapausten valinnassa voidaan minimoida.

Schürr ja muut [2010] perustivat FMT-mallin kielen enimmäkseen ominaisuusmalleihin, sillä niiden pääsovellusalue on ohjelmistotuotantolinjojen määrittely. He esittelevät alustavan metamallin FMT:stä, joka on hyvin UML-kaavion

tapainen. Abstraktina metaluokkana mallissa on sovellus, jonka uutena muuttujana on tietyn ominaisuuden muuttuvuuden *sidosajasta* (binding time) kertova ominaisuus, jota ominaisuusmalli tai CT-malli eivät tue. [Schürr *et al.* 2010] Sidosaika on se hetki, jolloin aliprosessi sisällytetään ohjelmistotuotteeseen [Pohl *et al.* 2005, 59]. Mallin muita luokkia ovat rajoitukset, ominaisuusryhmät, tyyppi, atominen ominaisuus ja testaus, joka jakaantuu testitapauksiin ja -ryhmiin. Testien ja sovellusten välistä yhteyttä käytetään kahteen tarkoitukseen. Testiryhmä yhdistää itsensä vaadittujen ominaisuuksien kanssa ja määrittelee missä tuotteissa tätä tiettyä testiryhmää voi käyttää. Lisäksi atomiset testit käyttävät tätä yhteyttä määritelläkseen tietyt tuotteet syöteparametrien kanssa. [Schürr *et al.* 2010]

Schürr ja muut [2010] käyttävät tutkimuksessaan auton ovelle suunniteltua tuotantolinjaa havainnollistamaan FMT-mallin käyttöä. FMT-mallin käyttöönotossa on neljä vaihetta (FMT-mallin valmistelu, testitapausten määrittely, tuotetapausten valinta ja testien toteutus). Mallin valmistelussa valitaan vaatimusmäärittelyn avulla ominaisuudet, jotka lisätään mallin. Perinteistä ominaisuusmallia voidaan käyttää FMT-mallin pohjana. Mahdolliset puuttuvat syöteparametrit lisätään malliin ja kaikille ominaisuuksille annetaan sidosaika. Auton oven yksi aliprosessi on turvavyö, jonka parametreja ovat kiinni ja auki. [Schürr *et al.* 2010]

Toisessa vaiheessa määritellään testitapaukset. Määrittelyssä käytetään pitkälti pohjana CT-mallia, mutta FMT-mallin tuomat lisäykset, kuten sidosajat, helpottavat prosessia. Testien määrittelyn jälkeen valitaan tuotetapaukset ja toteutetaan vaiheen kaksi testit. [Schürr *et al.* 2010] Havainnollistus ja mallin kuvaus ovat Schürrin ja muiden [2010] mukaan vielä yleisluontoisia, mutta he aikovat jalostaa niitä pitemmälle tulevissa tutkimuksissa.

FMT-malli on vielä kehityksen alla, mutta se tuntuu olevan hyvällä pohjalla. Yleisesti käytettyjen mallinnuksien yhdistäminen niin, että niiden hyvät puolet voitaisiin valjastaa sekä suunnittelun että testauksen käyttöön, on ideana hyvä. Myös Schürr ja muiden [2010] tutkimuksessaan tekemät havainnollistukset ovat vakuuttavia ja yksityiskohtaisia siihen nähden, että heidän mallinsa on vielä keskeneräinen. Uudet mallinnustyökalut ovatkin yksi tapa kehittää SPLE-paradigman testausta, sillä vanhat mallit voivat edesauttaa ongelmien pysymistä pinnalla. Ne eivät välttämättä pysy teknologian kehityksessä mukana, jos niitä ei kehitetä.

4.2. Evoluutioalgoritmit

Yksi raportoiduista SPLE-paradigman testauksen ongelmista on testitapausten määrä. Ensan ja muut [2012] ehdottavat ratkaisuna evoluutiopohjaisia geneettisiä algoritmeja, jotka automaattisesti luovat testattavien tapausten sarjat. Algoritmien tuottamia mahdollisia ratkaisuja Ensan ja muut [2012] kutsuvat kromosomeiksi ja jokainen kromosomi koostuu geeneistä. Jokainen geeni edustaa yhtä ominaisuutta.

Ensanin ja muiden [2012] ratkaisun hahmotelma sisältää viisi vaihetta. Vaiheessa 1 tuotelinjaa kuvaavasta ominaisuusmallista generoidaan satunnaisesti erilaisia ominaisuusmallikokoonpanoja ilman rajoituksia. Kokoonpanojen täytyy kuitenkin olla toimivia. Nämä kokoonpanot toimivat alkupopulaationa algoritmeille, jotka arvioivat vaiheessa 2 niiden hyvyyttä. Hyvyys määritellään *muuttuvuuden ja eheyden rajoitusten* (variability and integrity constraints) kautta. Vaiheessa 3 heikot kokoonpanot hylätään ja hyväksytyt kokoonpanot siirtyvät vaiheeseen 4, jossa niitä käytetään syötteenä *mutaatio- ja risteytysoperaatioille* (mutation and cross-over operations). Mutaatio-operaattori tuo kromosomiin ominaisuuden, jota ei ole sen edellisessä versiossa. Risteytys-operaatio taas muuttaa monia geenejä kromosomeissa. Vaiheessa 5 arvioidaan vielä kerran vaiheessa 4 kehitettyjä kokoonpanoja, sillä kokoonpanojen oikeellisuus pitää vielä varmistaa. Vaiheita 2-5 toistetaan, kunnes algoritmin lopetusehto täyttyy. Tutkimuksessa esitellään kaksi pääkriteeriä algoritmin lopetukselle: 1) maksimimäärä sukupolvia määritetään, jotta algoritmi ei jatkaisi ajallisesti loputtomiin, ja 2) peräkkäisten sukupolvien keskimääräinen hyvyys ei ylitä määrättyä tasoa, mikä tarkoittaa, että algoritmi ei pysty enää optimoimaan kokoonpanoja. Prosessin päätyttyä lopullinen populaatio määritellään testisarjaksi ja populaation kokoonpanot yksittäisiksi testeiksi. [Ensan *et al.* 2012]

Tutkijat testasivat malliaan vapaasti käytettävissä oleviin ominaisuusmalleihin Software Product Line Online Tools -sivuston kautta [Ensan *et al.* 2012]. Testatakseen kompromissia ominaisuuksien ja virheiden havainnoinnin välillä, Ensan ja muut [2012] tekivät useita testejä erisuuruisilla kokoonpanoilla. Testit osoittivat, että verrattuna tavallisiin ominaisuusmalleihin, joiden kompleksisuus on $O(2^n)$, Ensan ja muiden mallin kompleksisuus on vain $O(n)$, missä n on ominaisuusmallin yksi ominaisuus. He toteavat, että esitetty testisarjoja tuottava lähestymistapa pystyy kehittämään järkevän tasapainon ominaisuuksien ja virheiden havainnoinnin välille.

Ensan ja muut [2012] ovat selvittäneet hyvin yksityiskohtaisesti tutkimuksessaan lähestymistapaansa, joka näyttää myös käytännössä toimivan ainakin testatuilla ominaisuusmalleilla. Kuitenkin tämäkin tutkimus tarvitsee toistavaa

tutkimusta ja kirjoittajat itsekin toivovat, että muut tutkijat tekisivät tulevaisuudessa vertailevia tutkimuksia tästä lähestymistavasta [Ensan *et al.* 2012]. Tämä lähestymistapa on hyvin vakuuttava ja ainakin teoreettisesti täysin toimiva ratkaisu testitapausten vähentämiseen. Esitettyjen algoritmien jatkokehityksellä niitä voitaisiin ehkä soveltaa myös muihin testauksen osa-alueisiin.

4.3. Paritestauksen kehitys

CIT-strategia on suosittu lähestymistapa SPLE-paradigman testaukseen. Ominaisuuksien yhdisteleminen pareiksi (pairwise testing) on yksi strategian paljon huomiota saaneista aktiviteeteistä [Lopez-Herrejon *et al.* 2013]. Lopez-Herrejon ja muiden [2013] mukaan aiemmat tutkimukset ovat suhtautuneet paritestaukseen optimointiongelmana, jossa pääasiana on ollut joko kattavuus tai testisarjojen koko. Lopez-Herrejon ja muut [2013] kuitenkin käsittelevät näitä molempia, sekä kattavuutta että sarjojen kokoa, yhtä tärkeinä asioina. He esittelevät lineaarisen optimointiohjelman, jossa yritetään maksimoida testauksen kattavuutta sekä algoritmin, joka laskee Pareto-rintaman ominaisuusmallista. Tämä Pareto-rintama muodostuu siis niistä ratkaisuista, joissa kattavuus ja testisarjojen koko yhdistyy kombinaatioina, joissa toista tavoitetta ei voi parantaa heikentämättä toista. [Lopez-Herrejon *et al.* 2013]

Lopez-Herrejon ja muut [2013] testasivat lähestymistapaansa 118 ominaisuusmalliin, jotka ovat vapaasti käytettävissä. He pystyivät laskemaan jokaisen mallin Pareto-rintaman, mutta huomasivat, että algoritmin suoritusaika kasvaa nopeammin kuin lineaarisesti, kun tuotteiden määrä ominaisuusmallissa kasvaa. He toteavatkin, että jatkokehitys ja optimointi on tarpeen, jotta algoritmin suoritusaikaa saataisiin vähennettyä. Lisäksi he aikoivat vähentää turhia yhdistelmiä ja parantaa rajoituksia. [Lopez-Herrejon *et al.* 2013]

Vaikka Lopez-Herrejon ja muut [2013] huomasivatkin lähestymistavassaan vikoja, niin se vaikuttaa lupaavalta. Kahden CIT-strategialle tärkeän tavoitteen huomioonottaminen varmasti parantaisi SPLE-paradigman testauksen tuloksia niissä projekteissa, jotka tätä strategiaa käyttävät. Machado ja muut [2014] kuitenkin totesivat, että CIT on yksi suosituimpia testausstrategioita, joten sen jatkokehitys on varmasti hyödyllistä. Lopez-Herrejon ja muut [2013] nostivatkin esiin järkevän kysymyksen, sillä miksi ei tutkittaisi ja kehitettäisi lähestymistapoja, jotka ottaisivat sekä testauksen kattavuuden että testitapausten vähennyksen huomioon. Se tekisi jo suositusta strategiasta paremman ja auttaisi ratkaisemaan paradigman testauksen ongelmia. Jatkokehitys ja tutkimus tulee näyttämään tämänkin lähestymistavan pätevyyden.

4.4. Testitapausten priorisointi

Ensan ja muut [2011] ehdottavat kehittämäänsä testaustapausten priorisointimallia. Malli on kaksiosainen: ensimmäisessä vaiheessa testattavien ominaisuuksien määrää vähennetään valitsemalla ne ominaisuudet, jotka ovat tärkeimpiä ohjelman käyttötarkoitukselle sekä käyttäjille, ja toisessa vaiheessa jäljelle jäävien ominaisuuksien testaus priorisoidaan [Ensan *et al*. 2011]. Myös tämä priorisointi tehdään eniten haluttujen ominaisuuksien perusteella. He esittävät mallista havainnollistavan esimerkin, jossa esimerkkitapauksena on verkkokauppasovellus. [Ensan *et al.* 2011]

Tutkimuksessa näkyy lähestymistavan hyödyt, mutta metodia täytyy tutkia lisää empiirisesti. Ehdotus kuitenkin kuulostaa loogiselta, varsinkin kun tutkimuksissa on todettu, että kaikkien mahdollisten ominaisuuksien ja tilanteiden testaaminen on käytännössä mahdotonta. Esitetty malli pyrkiikin juuri Machadon ja muiden [2014] mainitsemaan ajankäytön vähennykseen saavuttamalla suurempi virheiden huomiointikyky. Näin Ensanin ja muiden [2011] ehdotus voi, varsinkin jatkotutkimuksen kautta, antaa toimivan vaihtoehdon korjaamaan testaukseen liittyviä ongelmia.

4.5. Dynaaminen SPL

Capilla ja muut [2014] käsittelevät dynaamisia ohjelmistotuoteperheitä (DSPL), jotka ovat syntyneet viime vuosina ohjelmistotuoteperheitä kohtaan jatkuvasti kasvavien vaatimusten johdosta. Tuoteperheiden pitäisi olla dynaamisempia ja DSPL-paradigma vastaa tähän haasteeseen Capilla ja muiden [2014] mukaan paremmin kuin perinteinen SPLE-paradigma. DSPL vastaa paremmin ajonaikaisiin muutosvaatimuksiin sekä sopii paremmin *mukautuville järjestelmille* (self-adaptive systems).

Capilla ja muut [2014] määrittelevät DSPL-paradigman viitekehyksessään paradigmaa käyttävien ohjelmistojen ominaisuuksien avulla, sillä heidän mukaansa paradigmalla ei ole vielä täysin vakiintunutta käytäntöä. He määrittelevät DSPL-paradigmalle kolme tunnusomaista ominaisuutta: 1) tuki ja hallinta ajonaikaiselle muuttuvuudelle, 2) moninainen ja dynaaminen muuttuvuuden sitominen ja 3) kontekstitietoisuus ja itsemukautuvuus itsenäiseen käytökseen. [Capilla *et al.* 2014]

DSPL on melko uusi paradigma, joten Capilla ja muut [2014] eivät pystyneet antamaan täysin päteviä ratkaisuja käsiteltyihin ongelmiin. Vaikka Capilla ja muut [2014] eivät varsinaisesti keskittyneet testaukseen, he mainitsivat ajonaikaiseen muuttuvuudenhallintaan liittyvän ajonaikaisen testauksen. Tämä tuo tietysti omat haasteensa DSPL-paradigmaan, mutta myös mahdollisuuksia.

DSPL vaikuttaa olevan pätevä vastine perinteiselle SPLE-paradigmalle ja on hyvä muistutus siitä, että teknologinen kehitys tuo mukanaan uusia paradigmoja, varsinkin jos vanhat paradigmat eivät pysy kehityksen perässä. Jos tulevaisuudessa DSPL-paradigman testaus osoittautuu toimivammaksi, niin SPLE voi jäädä taka-alalle. Voisin kuitenkin kuvitella, että samat ongelmat testauksessa ovat näkyvissä myös DSPL-paradigmassa, sillä paradigmat tuntuvat jakavan paljon samoja käytäntöjä. SPLE-paradigman ongelmia ei siksi tulisi vain hylätä ja jatkaa eteenpäin uudella käytännöllä, sillä ongelmat voi löytää edestään hyvin nopeasti uudestaan.

5. Keskustelu

Kirjallisuudessa on esitetty useita ongelmia ja ratkaisuja SPLE-paradigman testauksen ongelmiin. On melko outoa, että testauksen ongelmat ovat pysyneet vuosia, vaikka paradigmaa käyttäneet yritykset ja testejä kehittäneet tutkijat ovat varmasti huomanneet ongelmat, jotka ilmaantuvat ohjelmiston testausvaiheessa. Metzger ja Pohl [2014] toteavatkin, että SPLE-paradigman laaduntarkkailun ongelmat kuuluvat yksiin kauimmin avoinna olleista tutkimushaasteista ohjelmistokehityksen alalla. Engström ja Runeson [2011] antama mahdollinen selitys, eli tämän kaltaisen tutkimuksen kalleus ja suurimittaisuus, on todennäköisesti yksi syy ongelmien ratkaisemattomuuteen. Silti se, ettei testausta tutkita, voi aiheuttaa paljon suurempia kustannuksia itse paradigmaa implementoiviin yrityksiin kuin itse tutkimukseen. Siksi yrityksiä olisi hyvä saada mukaan tutkimustyöhön. Voi kuitenkin olla, että yritykset saattavat haluta pitää onnistuneet testaustavat omana tietonaan, sillä se antaa niille luonnollisesti kilpailuetua.

Yllättävintä on mielestäni se, että vaikka erilaisia testaustapoja on kyllä kehitetty ja tutkittu, niin kehitettyjä testaustapoja ei ole vertailtu keskenään. Käytäntö ei kuulosta kovin järkevältä, sillä vaikka tietty tapa testata tiettyä tuoteperhettä olisikin tehokas, niin se ei takaa tavan luotettavuutta muissa tilanteissa. Vertailevan tutkimuksen puutteen takia eri testausmetodeja ei ole kunnolla vertailtu toisiinsa ja näin ei olla saatu tutkittua tietyn tavan hyötyjä verrattuna toiseen. Tämä jättää suuren aukon SPLE-paradigman tutkimukseen. Pitää tietenkin muistaa, että moni tutkimuksen ongelmia laajemmin tutkinut julkaisu on ollut kirjallisuuskatsaus, jolloin on ollut mahdollista, että joitain tutkimuksia on jäänyt huomaamatta. Machado ja muut [2014] huomauttavatkin tästä tutkimuksessaan ja lisäävät, että vakiintuneen termistön puute puhuttaessa SPLE-paradigmasta vaikeuttaa täysin kuvaavan aineiston muodostamista. Ongelmien olemassaolo on kuitenkin todennäköistä, sillä myös Lee ja muut [2012] ja Engström ja Runeson [2011] raportoivat samankaltaisista ongelmista.

On varmasti monta syytä siihen, miksi eri testaustapoja ei ole tutkittu tarpeeksi, mutta tulevaisuudessa juuri uudelleen toistattavissa olevien empiiristen tutkimusten lisäys on tarvittavaa, jotta SPLE-paradigma pysyisi yrityksille kannattavana vaihtoehtona. Paradigman implementointi yrityksiin ei ole välttämättä suoraviivaista, ja se vaatii yritykseltä tiettyä organisaatiorakennetta [Pohl *et al.* 2005]. Jos paradigman käytöstä ei ole selkeää hyötyä, niin yritysten on parempi ottaa käyttöön jokin toinen paradigma. Pahimmassa tapauksessa tavallisten, yhdelle ohjelmistotuotteelle suunniteltujen käytäntöjen käyttö voi olla paras ratkaisu, jos SPLE ei tuo sen lupaamia hyötyjä. Testauksella on kuitenkin suuri rooli laadunvalvonnassa ja se voi viedä huomattavasti aikaa [Ensan *et al.* 2012].

Esitellyissä ratkaisuissa testauksen ongelmiin on ikävä kyllä havaittavissa samat ongelmat, kuin mistä kirjallisuudessa on puhuttu. Niistä kuitenkin osa vaikuttaa olevan päteviä ja mahdollisten jatkotutkimusten kautta esimerkiksi Ensan ja muiden [2012] ratkaisumalli voi osoittautua tehokkaaksi. Tämä on kuitenkin mahdollista vain, jos tutkijat itse jatkavat mallinsa kehitystä tai jos muu taho ottaa vastuun jatkosta itselleen.

Kuten mainitsin alussa, SPLE-paradigma ei ole ainut keino kehittää useita ohjelmistotuotteita, jotka jakavat samoja ominaisuuksia. Capilla ja muiden [2014] mainitsema dynaaminen SPLE-paradigma voi tulevaisuudessa vastata ohjelmistoihin kohdistuviin vaatimuksiin paremmin tai sitten se voi jäädä yhtä hyväksi vaihtoehdoksi perinteiselle SPLE-paradigmalle. Tulevaisuutta on vaikea ennustaa, mutta uusien vaihtoehtojen ilmestyessä SPLE-paradigman pitää uudistua tai antaa tietä uusille lähestymistavoille.

6. Yhteenveto

Tässä tutkielmassa suoritin kirjallisuusarvion Software product line engineering -paradigman testauksesta sekä testauksen ongelmista. Monet tutkimukset osoittivat samaan lopputulokseen: SPLE on suosittu paradigma, mutta sen testaukseen liittyvät käytännöt ovat monesti itse kehitettyjä, huonosti testattuja ja huonosti tutkittuja. Tulevaisuudessa vaaditaan enemmän empiiristä tutkimusta, jotta voitaisiin kehittää toimivia ja luotettavia testauskäytäntöjä. Erityisesti eri käytäntöjä vertaavia tutkimuksia kaivataan, jotta saataisiin selville kunkin lähestymistavan hyödyt suhteessa muihin.

Viiteluettelo

Rafael Capilla, Jan Bosch, Pablo Trinidad, Antonio Ruis-Cortés and Mike Hinchey. 2014. An overview of dynamic software product line architectures and techniques: Observations from research and industry. *The Journal of Systems and Software* 93, 3-23.

Liaping Chen and Muhammad Ali Babar. 2011. A systematic review of evaluation of variability management approaches in software product lines. *Information and Software Technology* 53, 4, 344–362.

Emelie Engström and Per Runeson. 2011. Software product line testing – A systematic mapping study. *Information and Software Technology* 53, 1, 2-13.

Alireza Ensan, Ebrahim Bagheri, Mohsen Asadi, Dragan Gasevic and Yevgen Biletskiy. 2011. Goal-Oriented Test Case Selection and Prioritization for Product Line Feature Models. In: *Proc. of the Eighth International Conference on Information Technology: New Generations (ITNG)* 291-298.

Faezeh Ensan, Ebrahim Bagheri and Dragan Gasevic. 2012. Advanced Information Systems Engineering. In: *Proc. of the 24th International Conference on Advanced Information Systems Engineering (CAiSE 2012)* 613-628.

Jihyun Lee, Sungwon Kang and Danhyung Lee. 2012. A Survey on Software Product Line Testing. In: *Proceedings of the 16th International Software Product Line Conference - Volume 1,* 31-40.

Roberto E. Lopez Herrejon, Francisco Chicano, Javier Ferrer, Alexander Egyed and Enrique Alba. 2013. Multi-objective Optimal Test Suite Computation for Software Product Line Pairwise Testing. In: *Proceedings of the 29th IEEE International Conference on Software Maintenance (ICSM)*, 404–407.

Ivan do Carmo Machado, John D. McGregor, Yguaratã Cerqueira Cavalcanti and Eduardo Santana de Almeida. 2014. On strategies for testing software product lines: A systematic literature review. *Information and Software Technology* 56, 10, 1183-1199.

Andreas Metzger and Klaus Pohl. 2014. Software product line engineering and variability management: achievements and challenges. In: *Proceedings of the on Future of Software Engineering,* 70-84.

Gilles Perrouin, Sebastian Oster, Sagar Sen, Jacques Klein, Benoit Baudry and Yves le Traon. 2012. Pairwise testing for software product lines: comparison of two approaches. *Software Quality Journal* 20, 3, 605-643.

Klaus Pohl, Günter Böckle and Frank van der Linden. 2005. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer Science and Business Media.

Andy Schürr, Sebastian Oster and Florian Markert. 2010. Model-Driven Software Product Line Testing: An Integrated Approach. In: *Proceedings of the 2010 Theory and Practice of Computer Science*, 112–131.

Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer and Gunter Saake. 2014. A classification and survey of analysis strategies for software product lines. *ACM Computing Surveys* 47, 1, 6:1-6:45.

K-means -klusterointialgoritmi ja sen variaatiot

Miikka Mäkipörhölä

Tiivistelmä.

K-means -klusterointialgoritmia hyödynnetään yleisesti klusterianalyysissa ja tiedonlouhinnassa. K-means -klusterointialgoritmin suurin ongelma on klustereiden määrän valitseminen ja niiden sijoittaminen. Tässä tutkielmassa perehdytään k-means -klusterointialgoritmiin ja muutamiin sen variaatioista. Tutkielmassa tarkastellaan myös miten näissä variaatioissa on pyritty kehittämään alkuperäisen k-means -klusterointialgoritmin tuloksia ja laskennallista nopeutta.

Avainsanat ja -sanonnat: K-means, K-medoids, K-means++, klusterointi, partitionaalinen klusterointi.

1. Johdanto

Tiedon klusterointia eli aineiston ryhmittelyä samankaltaisuuden perusteella voidaan hyödyntää monenlaisissa eri tehtävissä, yleisimmin kuitenkin klusterointia hyödynnetään tiedonlouhinnassa ja tilastollisissa analyyseissa. Klustereiden globaalin optimin löytäminen on laskennallisesti vaativaa, mutta k-means -klusterointialgoritmin kaltaiset heuristiset algoritmit konvergoituvat paikalliseen optimiin nopeasti.

K-means -klusterointialgoritmi, joka tunnetaan myös nimellä Lloydin algoritmi, on yksi tunnetuimmista ja vanhimmista klusterointialgoritmeista, Hartigan ja Wong [1979] esittelivät k-means -klusterointialgoritmin nykyisessä muodossaan jo vuonna 1979. Tuolloin esitetty muoto k-means -klusterointialgoritmista on vielä nykypäivänäkin yleisesti käytössä, vaikka algoritmista on tehty monia variaatioita vuosikymmenien aikana. Tässä tutkielmassa perehdytään tarkemmin kahteen eri muunnelmaan alkuperäisestä k-means -klusterointialgoritmista, k-means++ -klusterointialgoritmiin ja k-medoids -klusterointialgoritmiin. Tutkielmassa esitellään myös x-means, k-medians ja fuzzy c-means -klusterointialgoritmit, mutta näiden toimintaa ja toteutusta ei tarkastella tarkemmin.

Alkuperäisessä k-means -klusterointialgoritmissa on kaksi suurta ongelmaa: klustereiden aloitussijaintien ja klustereiden määrän valitseminen. Tämä tutkielma pyrkii analysoimaan, miten eri variaatiot k-means -klusterointialgoritmista ratkaisevat nämä ongelmat, ja miten tämä vaikuttaa niiden laskennalliseen nopeuteen ja tulosten tarkkuuteen.

Luvussa 2 määritellään aiheeseen liittyviä käsitteitä. Luvussa 3 esitellään tarkemmin k-means -klusterointialgoritmin ja sen variaatioiden toimintaperiaatteita. Luvussa 4 perehdytään tarkemmin mihin käyttötarkoituksiin k-means -klusterointialgoritmia voidaan hyödyntää. Luvussa 5 vertaillaan kokeellisesti k-means -klusterointialgoritmien ja muutaman sen variaation eroja esimerkkiaineistoja hyödyntämällä. Algoritmit, joita verrataan kokeellisesti, ovat alkuperäinen k-means, k-means++ ja k-medoids -klusterointialgoritmit. Luvussa 6 on koottu yhteen tutkielmassa tehdyt tutkielmassa tehdyt havainnot.

2. Käsitteitä

2.1 Klusterointi

Klusteroinniksi kutsutaan aineiston jakamista joukkoihin samankaltaisuuden perusteella. Useimmiten tavoitteena on pystyä muodostamaan aineiston luontaiset ryhmittymät, ilman ennakkotietoa siitä, miten aineistossa esiintyvät ryhmät muodostuvat. Klusterointialgoritmeja onkin useita ja jotkin algoritmit toimivat paremmin toisilla aineistoilla kuin toiset, johtuen niiden erilaisista tavoista muodostaa klustereita [Jain 2010]. Aineistoon sopiva klusterointialgoritmi tuottaa klustereita, joissa samaan klusteriin kuuluvien alkioiden eroavaisuus on mahdollisimman pieni, ja eri klustereihin kuuluvien alkioiden eroavaisuus on mahdollisimman suuri.

Klusterointialgoritmit voidaan jakaa hierarkkisiin ja partitionaalisiin. Hierarkkiset klusterointialgoritmit toimivat kahdella tavalla, joko jokainen alkio alkaa omasta klusterista ja näitä klustereita yhdistetään suuremmiksi klustereiksi, tai kaikki alkiot ovat alussa samassa klusterissa ja tätä klusteria jaetaan pienemmiksi klustereiksi. Jakamisen tai yhdistämisen perusteena käytetään alkioiden toisistaan eroavuutta, yleensä tätä mitataan pisteiden etäisyydellä toisistaan. Tuloksena hierarkkisista klusterointialgoritmeista saadaan klusteripuu, josta nähdään miten klusterit liittyvät toisiinsa.

Partitionaaliset klusterointialgoritmit, joihin k-means -klusterointialgoritmi kuuluu, jakavat alkiot toisiinsa liittymättömiin klustereihin, niin että jokainen alkio kuuluu vain yhteen klusteriin. Partitionaaliset klusterointialgoritmit ovat hierarkkisia klusterointialgoritmeja käytetympiä monilla tieteenaloilla, käytettävissä olevan datan muodosta johtuen. Ennen klusteroinnin suorittamista on myös yleistä vähentää aineistossa olevien piirteiden lukumäärää, jotta klusterointi olisi nopeampaa. Tämä voidaan tehdä algoritmisesti, jolloin aineistosta hyödynnetään vain tärkeimmät piirteet. Pääkomponenttianalyysi on muun muassa klusterointialgoritmien yhteydessä yleisesti käytetty menetelmä piirteiden

lukumäärän vähentämiseen. K-means -klusterointialgoritmi on tunnetuin ja yksinkertaisin partitionaalinen klusterointialgoritmi [Jain 2010].

Klusterointia voidaan hyödyntää moniin eri käyttötarkoituksiin, esimerkiksi kuvan segmentointi voidaan ratkaista klusterointialgoritmeilla [Ray and Turi 1999]. Yritykset voivat hyödyntää klusterointialgoritmeja muun muassa markkinatutkimuksissa [Punj and Stewart 1983] ja suosittelujärjestelmissä [Kim and Ahn 2007].

2.2 Euklidinen etäisyys & Manhattan-etäisyys

Euklidinen etäisyys on kahden pisteen välinen etäisyys ns. suoraa linjaa pitkin. Euklidinen etäisyys tasossa voidaan laskea Pythagoraan lauseella:

$$d(x,y) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} .$$

Euklidista etäisyyttä käytetään k-means -klusterointialgoritmissa klusterin ja alkion välisen etäisyyden laskemiseen. Euklidinen etäisyys voidaan myös laskea korkeampiulotteisessa avaruudessa; tällöin kaava on muodossa

$$d(x,y) = \|x - y\|_2 = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2} .$$

Manhattan-etäisyys on yhteen laskettu absoluuttinen ero kahden pisteen arvojen välillä. Manhattan-etäisyys kahden pisteen välillä *n*-ulotteisessa avaruudessa voidaan laskea kaavalla

$$d(x,y) = \|x - y\|_1 = \sum_{i=1}^{n} |x_i - y_i| .$$

2.3 Globaali ja paikallinen optimi

Paikallinen optimi on ratkaisu, jonka naapurustosta ei löydy parempaa ratkaisua. K-means -klusterointialgoritmin tapauksessa tämä tarkoittaa parasta mahdollista ratkaisua, joka voidaan saada klustereiden aloitussijaintien ja määrän sisällä. Globaali optimi on paras mahdollinen ratkaisu kaikkien ratkaisujen joukosta. Jotta voidaan varmistua että globaali optimin löytymisestä, jokainen mahdollinen ratkaisu pitää käydä läpi, joka on yleensä laskennallisesti erittäin raskas toimenpide.

3. Algoritmit

3.1 K-means

K-means -klusterointialgoritmin toimintaperiaate on erittäin yksinkertainen, algoritmi pyrkii luokittelemaan *n* tietoalkioita *k*:hon klusteriin, jossa jokainen tietoalkio kuuluu lähimpään klusteriin. K-means -klusterointialgoritmin pseudokoodi (ks. algoritmi 1) ei sisällä klustereiden aloitussijaintien asettamista, koska siihen voidaan hyödyntää monia eri algoritmeja.

```
1 # Toistetaan algoritmia kunnes yksikään alkio n ei vahda klusteria k
2 while(True):
3     # Merkitään jos jokin alkio on vaihtanut klusteria
4     changes = False
5
6     # Tyhjätään jokaisen klusterin k alkio lista
7     for k in clusters:
8         k.nodes = []
9
10    # Haetaan uusin lähin klusteri jokaiselle alkiolle n
11    for n in data:
12        new_cluster = n.find_nearest_cluster(clusters)
13        # Tarkistetaan onko lähin klusteri k vaihtunut
14        if new_cluster is not n.cluster:
15            n.cluster = new_cluster
16            changes = True
17        # Lisätään alkio lähimmän klusterin alkio listaan
18        n.cluster.nodes.append(node)
19
20    # Jos yksikään alkio n ei vaihtanut klusteria k voidaan lopettaa
21    if changes is False:
22        break
23
24    # Lasketaan uusi sijainti jokaiselle klusterille k
25    for k in clusters:
26        mean_axes = []
27        new_location = []
28
29        # Lisätään lista jokaista ulottuvuutta vasten listaan
30        for dimension in data.coordinates:
31            mean_axes.append([])
32
33        # Lisätään listoihin pisteiden koordinaatit
34        for n in cluster.nodes:
35            for dim in range(n.coordinates):
36                mean_axes[dim].append(node.coordinates[dim])
37        # Lasketaan klusterin uusi painopiste
38        for axis in mean_axes:
39            new_location.append(mean(axis)
```

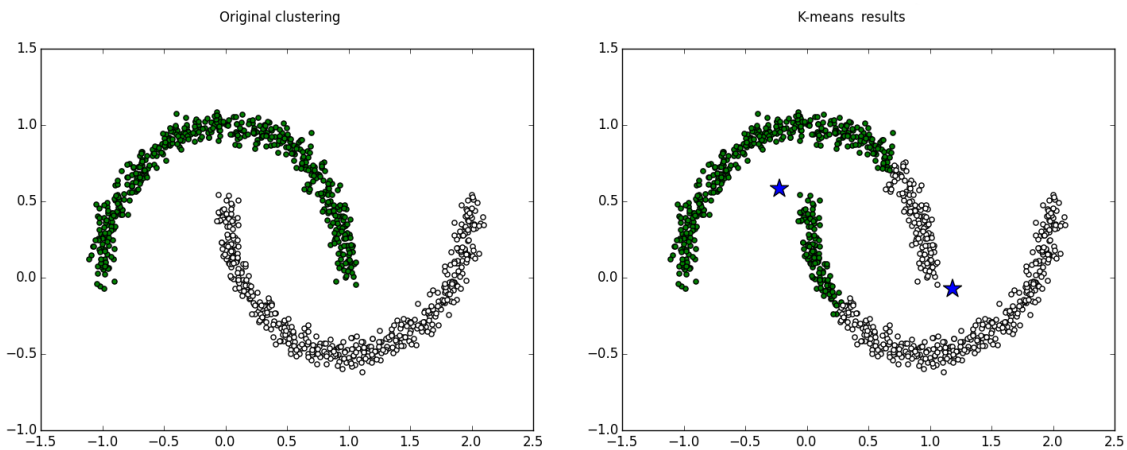
```
41          k.coordinates = new_cluster.coords
```

Algoritmi 1. K-means -klusterointialgoritmin pseudokoodi

K-means -klusterointialgoritmissa on kaksi vaihetta: sijoitus- ja päivitysvaihe. Sijoitusvaiheessa jokainen tietoalkio asetetaan euklidisen etäisyyden mukaan lähimpään klusteriin. Päivitysvaiheessa klustereille lasketaan uudet painopisteet, jotka määrittävät klusterien uudet sijainnit. Algoritmi on saavuttanut päätepisteen, kun yksikään tietoalkio ei vaihda klusteria sijoitusvaiheessa; tätä tilannetta kutsutaan konvergoitumiseksi [Hartigan and Wong 1979].

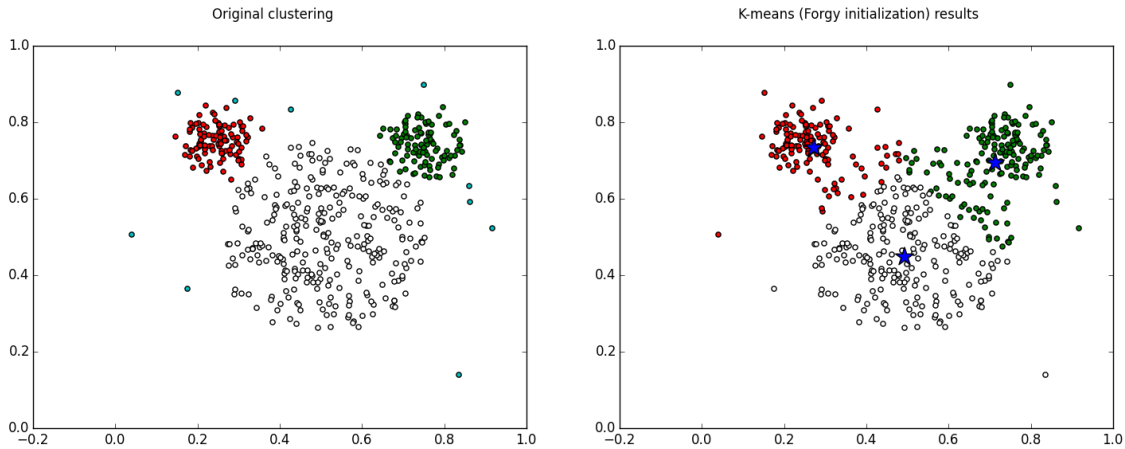
Klustereiden aloitussijaintien määrittämiseen on lukuisia eri vaihtoehtoja, mutta yleisimmin käytetyt vaihtoehdot ovat satunnainen sijoitus ja Forgyn metodi [Hamerly and Elkan 2002]. Forgyn metodissa aineistosta valitaan satunnaiset alkiot klustereiden keskipisteiksi. Satunnaisessa sijoituksessa jokainen alkio asetetaan satunnaiseen klusteriin ja klusterin aloitussijainti on kaikkien siihen kuuluvien alkioiden sijaintien keskiarvo.

K-means -klusterointialgoritmi konvergoituu aina paikalliseen optimiin, mutta algoritmi ei anna takeita globaalin optimin löytämisestä. On kuitenkin osoitettu että k-means algoritmi konvergoituu globaalin optimiin, jos klusterit ovat aineistossa vahvasti erillään [Meila 2006]. Tästä syystä algoritmi yleensä suoritetaankin useampaan kertaan, jotta voidaan löytää paras mahdollinen tulos. Jos tiedossa ei ole, kuinka monta eri klusteria käytetty data sisältää, testataan algoritmia yleensä eri määrillä klustereita. Klusterien alkusijainnit ja klusterien määrä vaikuttavat k-means -klusterointialgoritmin tuloksiin huomattavasti, ja tästä syystä monet variaatiot k-means -klusterointialgoritmista vaikuttavatkin näihin kahteen tekijään.



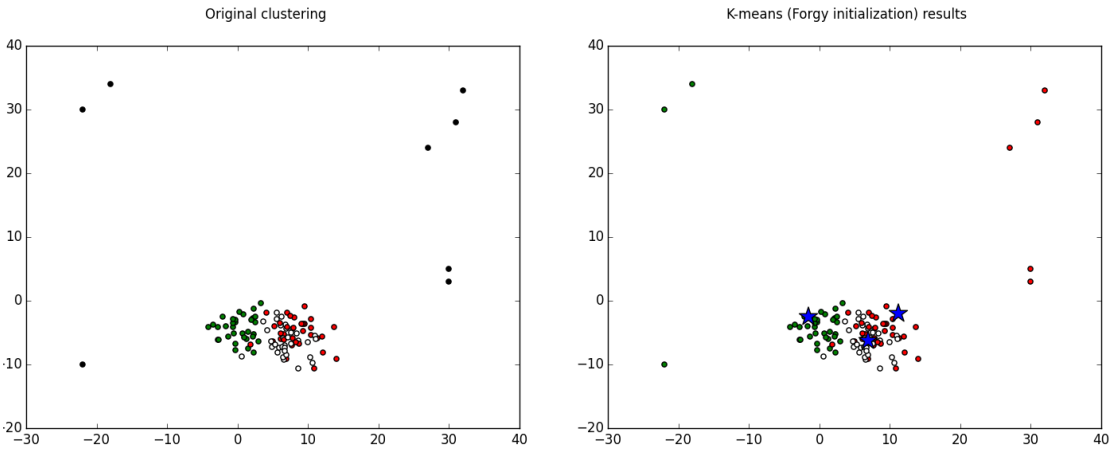
Kuva 1. K-means -klusterointialgoritmi, pyöreät klusterit esimerkki

K-means -klusterointialgoritmissa on useita heikkouksia, jotka saattavat esiintyä riippuen aineiston laadusta. Yleisin ongelma on klustereiden muoto (ks. kuva 1), k-means -klusterointialgoritmin tuottamat klusterit ovat aina pyöreitä, johtuen klustereiden sijaintien laskutavasta (alkioiden sijaintien keskiarvo on klusterin keskipisteen sijainti), tästä syystä algoritmi ei yleensä tuota hyviä klustereita aineistoissa, jossa klusterit eivät ole pyöreän muotoisia ja tarpeeksi erillään toisistaan. Toinen ongelma on klustereiden tiheys (ks. kuva 2), k-means -klusterointialgoritmi pyrkii aina tuottamaan klustereita, joissa on lähes yhtä monta alkiota.



Kuva 2. K-means -algoritmi, klusterien tiheydet [Schubert *et al*. 2015]

Kolmantena ongelmana on algoritmin kyvyttömyys käsitellä aineistossa esiintyviä suuresti muista poikkeavia alkioita(ks. kuva 3), koska klusterin sijainti lasketaan siihen kuuluvien alkioiden etäisyyden keskiarvolla, yksittäiset suuresti poikkeavat arvot aineistossa voivat vaikuttaa klustereiden sijaintiin huomattavasti.



Kuva 3. K-means -klusterointialgoritmi, poikkeavat alkiot

K-means -klusterointialgoritmi on verrattain kevyt laskennallisesti tavanomaisilla aineistoilla, joten useiden iteraatioiden laskeminen eri klusterien aloitussijanneilla ei ole raskasta verrattuna moniin muihin algoritmeihin. Suurilla aineistoilla k-means -algoritmin suoritusajat kasvavat suuriksi, rinnakkaislaskennalla voidaan kuitenkin helpottaa tätä ongelmaa, mutta aineiston kasvaessa ”Big Data”-mittakaavoihin klusterointi on suhteellisen raskasta, riippumatta toteutustavasta. Lloydin esittämän heuristisen k-means -klusterointialgoritmin aikavaatimus annetaan yleensä muodossa $O(nkdI)$ [Manning *et al*. 2008], jossa n on alkioiden lukumäärä, k on klustereiden lukumäärä, d on aineistossa olevien piirteiden lukumäärä ja I on iteraatioiden lukumäärä. Aineistossa, jolla on klusteroituva rakenne, k-means -algoritmin aikavaatimus voidaan luokitella yleensä lineaariseksi nopean konvergoitumisen johdosta. Arthur ja Vassilvitskii [2006] esittivät kuitenkin, että huonoimmassa tapauksessa k-means -klusterointialgoritmin aikavaatimus on superpolynominen.

3.2 K-means++

K-means++ -algoritmi on Arthurin ja Vassilvitskiin [2007] esittämä kehittyneempi versio k-means -klusterointialgoritmista. K-means++ -klusterointialgoritmi lisää k-means -klusterointialgoritmiin klusterien aloitussijaintien määrittävän algoritmin. K-means++ -klusterointialgoritmi saavuttaakin yleensä noin kaksi kertaa nopeammin konvergoitumisen kuin k-means -klusterointialgoritmi [Arthur and Vassilvitskiin 2007].

K-means++ -klusterointialgoritmin aloitussijaintien määrittävä algoritmi on yksinkertainen, ja se perustuu ideaan että klusterien aloitussijainnit kannattaa hajauttaa toisistaan (ks. algoritmi 2).

```
1 # Valitaan satunnaisesti yksi alkio ensimmäisen klusterin sijainniksi
2 cluster.append(Random(data))
3 cluster_count = 1
4
5 # Toistetaan kunnes klustereita on valittu määrä k
6 while cluster_count is less than k:
7       # Käytetään klusterin valinnan todennäköisyyteen
8       sum_distances = 0
9
10      # Lasketaan jokaiselle alkiolle etäisyys lähimpään klusteriin
11      for n in data:
12              n.distance_ncluster(clusters)
13              # Lasketaan yhteen alkioiden etäisyydet toisessa potensissa
14              sum_distances = sum_distances + n.distance_ncluster ^ 2
15
16      # Valitaan painotetulla todennäköisyydellä uusi alkio klusteriksi
17      # Painotus perustuu alkion etäisyyteen lähimmästä klusterista k
18      for n in data:
19              if choose_new((n.distance_ncluster ^ 2), sum_distances):
20                      cluster.append(n)
21                      cluster_count = cluster_count + 1
22                      # Ei käydä läpi enempää alkioita jos klusteri valittiin
23                      break
```

Algoritmi 2. K-means++ -algoritmin pseudokoodi

Klustereiden aloitussijaintien määrityksen jälkeen k-means++ -klusterointialgoritmi etenee samalla tavalla kuin k-means -klusterointialgoritmi. K-means++ -algoritmi parantaa myös klustereiden määrityksen tarkkuutta, Vassilvitskiin ja Arthurin [2007] testeissä k-means++ -algoritmin löytämien klusterien tarkkuus oli vähintään 10 % parempi kuin k-means -klusterointialgoritmin.

K-means|| -algoritmi on kehittyneempi versio k-means++ -algoritmista. Sen suurin etu k-means++ -klusterointialgoritmiin on sen rinnakkaislaskettavuus ja skaalautuvuus. K-means|| ei vaadi *k*-määrää alkioiden läpikäymistä, jotta klustereiden aloitussijainnit voidaan laskea, ja on tästä syystä laskennallisesti nopeampi, jos aineiston koko on erittäin suuri. [Bahmani *et al*. 2012]

3.3 K-medoids

Kaufman ja Rousseeuw [1987] esittelivät k-medoids -klusterointialgoritmin, nimeltä PAM (Partition Around Medoids), jonka toiminta perustuu siihen, että klustereiden keskipisteinä toimivat aineistosta poimitut alkiot, joita kutsutaan medoideiksi, ja alkioiden erovaisuutta mitataan joko Manhattan-etäisyydellä, tai euklidisella etäisyydellä.

Park ja Jun [2009] esittelivät uuden version k-medoids -klusterointialgoritmista, joka laskee alkiosta etäisyysmatriisin ja hyödyntää sitä ensimmäisten klusterien sijoittamiseen. Etäisyysmatriisia hyödynnetään myös algoritmin päivitysvaiheessa, jolloin lasketaan siirtyvätkö klusterien keskipisteet uusiin sijainteihin. Klustereiden aloitussijainnit määritetään myös algoritmisesti, toisin kuin PAM Parkin ja Junin [2009] esittelemä k-medoids -klusterointialgoritmi tuottaa parempia tuloksia kuin k-means -klusterointialgoritmi, ja on laskennallisesti huomattavasti kevyempi kuin Kaufmanin ja Rousseeuwn [1987] esittelemä PAM.

K-medoids sisältää myös klusterien aloitussijainnit valitsevan algoritmin. Medoidien, eli klustereiden keskipisteiden, aloitussijainnit valitaan alkioiden määrän ja läheisyyden perusteella. Jokaiselle alkiolle lasketaan arvo v_j, joka saadaan kaavalla

$$v_j = \sum_{i=1}^{n} \frac{d_{ij}}{\sum_{l=1}^{n} d_{il}}, j = 1, \ ..., n \quad ,$$

jossa d_{ij} on alkion i etäisyys alkiosta j, d_{il} on alkion i etäisyys alkiosta l. Alkioiden etäisyys toisistaan lasketaan euklidisen etäisyyden kaavalla. Medoidien aloitussijainneiksi valitaan alkiot, joiden v_j arvo on suurin. Aloitussijainnit ovat myös aina samat samalla aineistolla, koska kaava ei sisällä minkäänlaista valintatodennäköisyyttä medoideille [Park and Jun 2009].

K-medoids -algoritmin medoidien painopisteen laskenta perustuu hieman erilaiseen periaatteeseen kuin k-means -algoritmin. Medoidien uusi keskipiste haetaan medoidiin kuuluvien alkioiden sisältä, valintaperusteena käytetään alkion muihin medoidin sisällä olevien alkioiden etäisyyksien summaa. Alkio, jolla tämä etäisyyksien summa on pienin valitaan medoidin uudeksi painopisteeksi. K-medoids -klusterointialgoritmi on konvergoitunut, kun kaikkien edellä mainittujen medoidien etäisyyksien summien summa ei ole pienempi kuin edellisellä iteraatiolla (ks. algoritmi 3).

```
1 # Toistetaan kunnes medoidien etäisyydet eivät pienene
2 while sum_distances < old_sum_distances
3       # Otetaan talteen viime kierroksen summatut etäisyydet
4       old_sum_distances = sum_distances
5
6       # Haetaan jokaiselle klusterille uusi medoidi
7       for k in clusters:
8               minimum = null
9               new_medoid = null
10
11              # Etsitään alkio joka minimoi etäisyyden muihin alkiohin
12              for n in k.nodes
13                      sum_dist = 0
14                      for l in k.nodes
15                              sum_distances += distance_matrix[n][l]
16                      if sum_distances < minimum
17                              new_medoid = n
18                              minimum = sum_distances
19              k.medoid = new_medoid
20
21      # tyhjätään klustereiden alkio lista
22      clusters.empty_nodes()
23
24      # Asetetaan jokainen alkio lähimpään medoidiin
25      for n in nodes
26              new_cluster = node.find_nearest_cluster(clusters)
27              if new_cluster != node.cluster
28                      node.cluster = new_cluster
29              node.cluster.nodes.add(node)
30
31      # Lasketaan uusi etäisyyksien summa medoideista alkiohin
32      for k in clusters
33              k.update_sum_distances()
34              sum_distances += k.sum_distance
```

Algoritmi 3. K-medoids -klusterointialgoritmin pseudokoodi

K-medoidsin toiminnassa esiintyy samat ongelmat klustereiden muodon ja tiheyden kanssa kuin k-means -klusterointialgoritmissa, mutta etäisyysmatriisin laskeminen vaikeuttaa k-medoidsin toimintaa suurien aineistojen klusteroinnissa. Etäisyysmatriisin muodostaminen on aina aikavaatimukseltaan $O(n^2)$, ja etäisyysmatriisin säilyttäminen muistissa on myös muistivaatimukseltaan $O(n^2)$. Nämä kaksi tekijää tekevät suurien aineistojen laskemisesta huomattavan paljon raskaampaa k-medoids -algoritmilla kuin k-means -algoritmilla.

3.4 X-means, k-medians ja fuzzy C-means

X-means klusterointialgoritmi perustuu myös samankaltaiseen alkuparametrien optimointiin kuin k-means++, mutta x-means -algoritmi optimoi klusterien lukumäärää. X-means klusterointialgoritmissa klusterien määrää ei määritetä tarkasti vaan algoritmille voidaan syöttää arvoväli, jossa sopiva klusterien määrän epäillään sijaitsevan. Algoritmien kehittäjien testeissä x-means -klusterointialgoritmi konvergoitui vähintään kaksi kertaa nopeammin, kuin tavanomainen k-means -klusterointialgoritmi. [Pelleg and Moore 2000]

K-medians on samankaltainen klusterointialgoritmi kuin k-means -klusterointialgoritmi, mutta klusterin painopisteen sijoittamiseen käytetään klusteriin kuuluvien alkioiden mediaani. K-medians algoritmissa alkioiden eroavaisuutta lasketaan Manhattan-etäisyydellä. K-medians -klusterointialgoritmia ei pidä sekoittaa k-medoids -klusterointialgoritmiin, koska k-medians -klusterointialgoritmissa saatu klusterin keskipiste ei sijaitse välttämättä alkuperäisessä aineistossa, toisinkuin k-medoids -klusterointialgoritmissa, jossa medoidina voi esiintyä vain alkuperäisessä aineistossa oleva alkio. [Bradley *et al*. 1997]

Fuzzy C-means -klusterointialgoritmissa alkioiden kuulumista klustereihin mitataan jäsenyysfunktiolla, joka voi saada arvoja väliltä nolla ja yksi. Tätä jäsenyysarvoa käytetään klusterin uuden painopisteen määrittämiseen, jossa klusterin painopiste on klusterin kaikkien alkioiden eroavaisuuden keskiarvo painotettuna jäsenyysfunktion arvolla. Jäsenyysfunktiota ja klusterin painopisteen laskutapaa lukuun ottamatta fuzzy C-means -klusterointialgoritmi toimii lähes samalla tavalla kuin k-means -klusterointialgoritmi. [Bezdek *et al*. 1984]

4. Käyttökohteita

K-means -klusterointialgoritmia hyödynnetään monilla eri tieteenaloilla, johtuen pääosin sen yksinkertaisuudesta, mutta myös sen tehokkuudesta. Perehdymme lyhyesti kolmeen käyttökohteeseen, joissa voidaan hyödyntää k-means -klusterointialgoritmia: konenäkö, koneoppiminen ja suosittelujärjestelmät.

4.1 Konenäkö

K-means -klusterointialgoritmia voidaan käyttää konenäön tukena, erityisesti kuvan segmentoinnissa. Kuvan segmentoinnissa kuvan pikselit jaetaan pikselijoukkoihin ja päämääränä on muodostaa yksinkertaisempi versio alkuperäisestä kuvasta, analysoinnin helpottamiseksi. Kuvan segmentoinnissa klustereiden eroavaisuuden merkitsijänä hyödynnetään yleensä joko alkioiden väriarvoja

[Ray and Turi 1999] tai valovoimaisuutta [Pham *et al*. 2006]. K-means -klusterointialgoritmia käytetäänkin kuvan segmentoinnissa yleensä jonkin muun algoritmin kanssa, jotta voidaan tehdä johtopäätöksiä saaduista klustereista automaattisesti. Ng ja muut [2006] esittelivät keinon hyödyntää k-means -klusterointialgortimia lääketieteellisten kuvien segmentoinnin esiprosessoinnissa.

K-means -klusterointialgoritmia voidaan myös hyödyntää kuvan värien kvantisointiin. Värien kvantisointi toteutetaan klusteroimalla kuvan pikselit *k*-määrään klustereita, jonka jälkeen jokaisen klusterin alkion väri muutetaan samaksi kuin klusterin, joka on lähimpänä kyseistä alkiota. Kuvassa on jäljellä tämän prosessin jälkeen *k* väriä. Kvantisointi voidaan myös suorittaa ennen kuvan segmentointia, segmentoinnin helpottamiseksi.

4.2 Koneoppiminen ja tiedonlouhinta

Koneoppiminen ja tiedonlouhinta ovat erittäin läheisiä käsitteitä, ja tiedonlouhinnassa käytetäänkin yleensä samoja koneoppimisalgoritmeja. Koneoppimisen tavoitteena on oppia ja tehdä johtopäätöksiä tiedosta algoritmisesti, kun taas tiedonlouhinnassa pyrkimyksenä on pikemminkin kerätystä tiedosta uusien mallien löytäminen. Koneoppiminen voidaan jakaa kahteen tärkeimpään osa-alueeseen: ohjattu oppiminen ja ohjaamaton oppiminen. Ohjatussa oppimisessa algoritmi tarvitsee oppimisaineiston tai opettajan, jonka avulla ohjatun oppimisen algoritmi pystyy laskemaan tilastollisesti, minkälaisia tuloksia aineistosta tuotetaan. Ohjatun oppimisen alle voidaan laskea myös puoliohjattu oppiminen, jossa algoritmille syötetty opetusaineisto ei sisällä kaikkia oikeita tuloksia. Ohjaamaton oppiminen, jonka alle k-means -klusterointialgoritmi voidaan lukea, on tiedosta johtopäätösten tekoa, ilman ennakkotietoa alkuperäisen aineiston merkityksestä tai piirteistä. K-means -klusterointialgoritmia voidaan myös hyödyntää piirreoppimisessa, joko ohjaamatonta oppimista tai ohjattua oppimista varten [Coates and Andrew 2012].

4.3 Suosittelujärjestelmät

Suosittelujärjestelmät ovat nykyään tärkeä osa verkkokauppojen toimintaperiaatetta. Asiakkaille suositellaan tuotteita heistä kerätyn tiedon perusteella tai samalla maantieteellisellä alueella sijaitsevien ostoksia tehneiden asiakkaiden ostosten perusteella. Suosittelujärjestelmät pohjautuvatkin yleensä jonkinlaiseen asiakkaiden ryhmittelyyn heidän ostokäyttäytymisensä perusteella, johon k-means -klusterointialgoritmi soveltuu hyvin.

Kim ja Ahn [2007] esittelivät muunnellun k-means -klusterointialgoritmin, jota he kutsuivat nimeltä: GA k-means -klusterointialgoritmi. Esiteltyä algoritmia, itsejärjestäytyviä karttoja ja alkuperäistä k-means -klusterointialgoritmia vertailtiin kokeellisesti hyödyntämällä oikean verkkokaupan asiakasaineistoa. Saatujen tulosten perusteella GA k-means -klusterointialgoritmi tuotti vertailluista algoritmeista parhaat tulokset ja algoritmin tuottamat tuotesuositukset olivat asiakkaiden mielipidemittausten mukaan hyviä. [Kim and Ahn 2007]

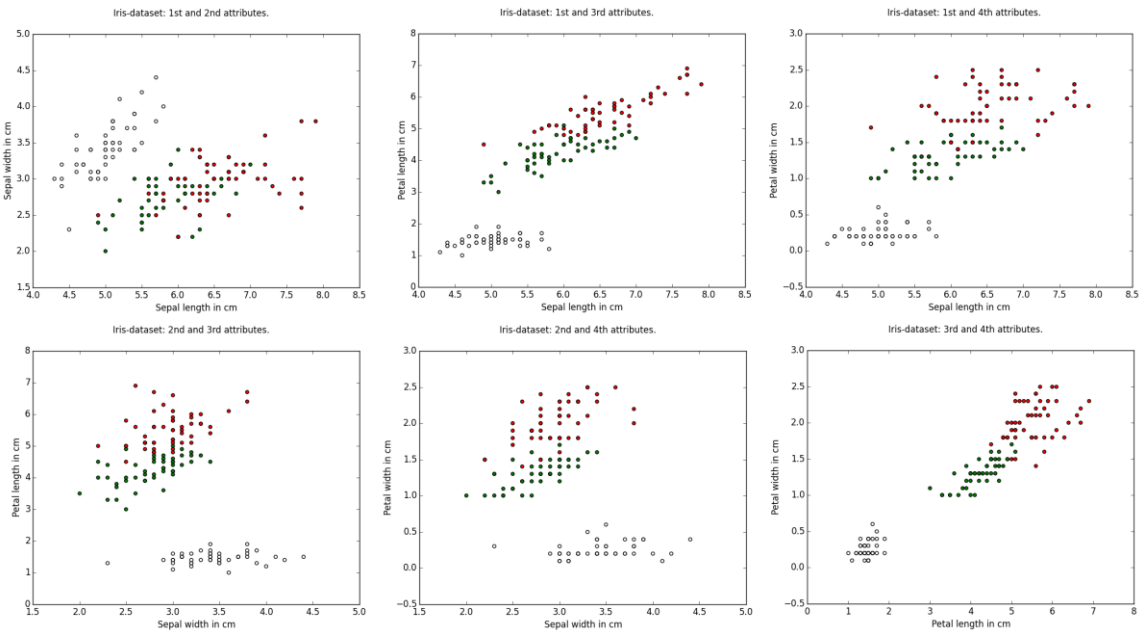
5. Algoritmien kokeellinen vertailu

Algoritmien vertaileminen testiaineistoilla helpottaa ymmärtämään miten algoritmien eri aikavaatimukset ja toimintaperiaatteet esiintyvät käytännön toteutuksissa. Vertailemme k-means, k-means++ ja k-medoids -algoritmien nopeutta, vaadittavien iteraatioiden määrää konvergenssiin ja luokittelutarkkuutta.

5.1 Toteutus

Algoritmit toteutettiin Python -ohjelmointikielellä. Toteutetut algoritmit ovat seuraavat: k-means, k-means++ ja k-medoids -klusterointialgoritmit. K-medoids -klusterointialgoritmista toteutettiin kaksi eri variaatiota, toisessa variaatiossa algoritmin hyödyntämä etäisyysmatriisi lasketaan jokaisella ajokerralla ja huomioidaan algoritmin ajoajassa, toisessa etäisyysmatriisi lasketaan erillään muusta algoritmista ja sitä ei huomioida keskimääräisessä ajoajassa. Toteutuksissa hyödynnettiin suosittua scikit-learn python -koneoppimiskirjastoa [Pedregosa *et al*. 2011].

Scikit-learn -kirjastossa on valmis toteutus k-means ja k-means++ -klusterointialgoritmeille, mutta näitä valmiita toteutuksia ei hyödynnetty. Scikit-learn -koneoppimiskirjastoa hyödynnettiin testiaineistojen generoimiseen ja Iris-aineiston lataamiseen. Iris-aineisto [Lichman 2013] on yksi tunnetuimmista aineistoista, ja sitä käytetään usein luokittelualgoritmien esittelyssä. Se sisältää kolme luokkaa, ja kaksi näistä luokista ei ole lineaarisesti erotettavissa toisistaan (ks. kuva 4), jokaisella alkiolla on 4 eri piirrettä ja luokkamuuttuja. Generoidut aineistot luotiin satunnaisesti, mutta generoinnissa hyödynnettiin siemenarvoa pseudosatunnaisen lukugeneraattorin alustamiseksi, jotta suoritetut testit olisi helppo toistaa. Kaikki tutkielman kuvat on toteutettu hyödyntäen matplotlib-kirjastoa [Hunter 2007]. Kuvaajat on piirretty aineistojen kahta ensimmäistä piirrettä käyttäen.



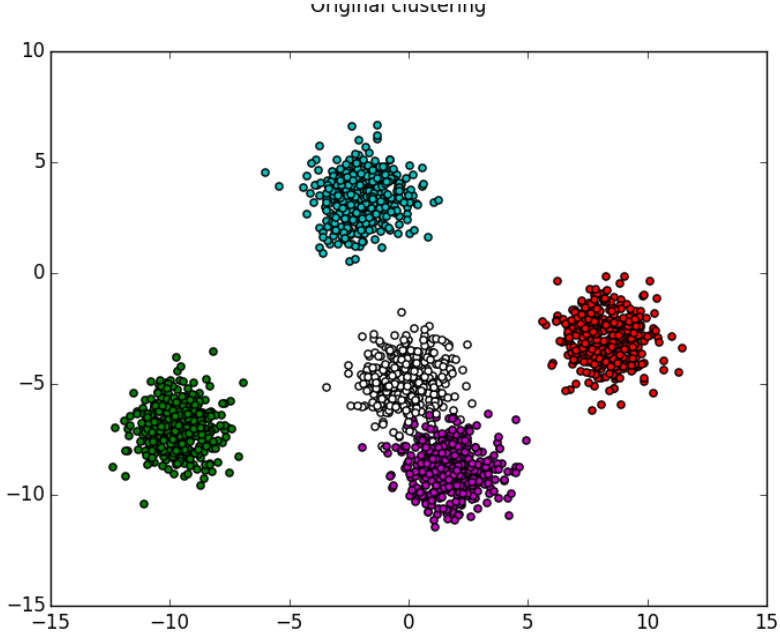
Kuva 4. Iris-aineisto

5.2 Algoritmien ajoaikojen vertailu ja konvergoitumisnopeuden vertailu

Algoritmien ajoaikojen ja konvergoitumisnopeuden vertailua varten generoitu aineisto sisälsi 2000 alkiota, jokaisella alkiolla oli 5 piirrettä ja luokkamuuttuja, klustereiden keskihajonta oli 1 ja aineistossa oli yhteensä 5 klusteria (ks. kuva 5). Jokainen algoritmi suoritettiin 100 kertaa, ja algoritmin ajoaikaa kuvaamaan valittiin näiden suoritusaikojen keskiarvo. Vertailu suoritettiin myös Iris-aineistolla (ks. kuva 4).

Algoritmi	km. ajoaika (ms)	km. iteraatioiden lukumäärä (i)
K-means (Forgy)	1112	12.63
K-means++	604	3.65
K-medoids	5897	3
K-medoids (ennalta laskettu etäisyysmatriisi)	5827	3

Taulukko 1. Algoritmien ajoaikojen tulokset generoidulla aineistolla



Kuva 5. Algoritmien ajoaikojen vertailussa käytetty generoitu aineisto

K-medoidsin iteraatioiden määrä konvergenssin saavuttamiseksi on aina kolme generoidulla aineistolla ja Iris-aineistolla, johtuen algoritmin medoidien aloitussijainnit määrittävästä algoritmista, joka ei sisällä satunnaista tekijää. Generoidulla aineistolla suoritetuissa testeissä (ks. taulukko 1) k-means++ on selvästi nopeampi kuin vertailun muut algoritmit. K-medoids -klusterointialgoritmin erilainen toimintaperiaate hidastaa algoritmia generoidulla aineistoilla, vaikka algoritmi saavuttaakin konvergenssin jo kolmen iteraation jälkeen. Iris-aineistolla k-medoidsin nopeus on jo lähempänä k-means -klusterointialgoritmia, mutta on vielä kuitenkin noin kaksi kertaa hitaampi.

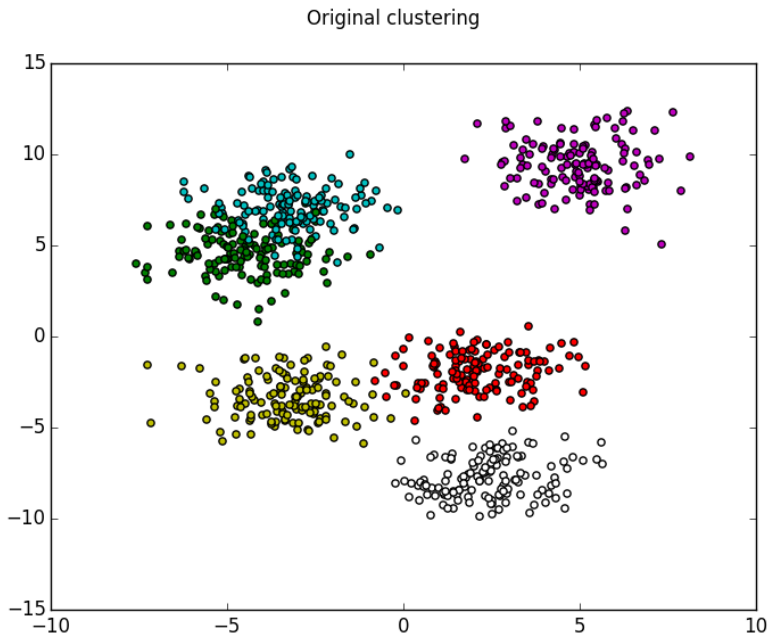
K-medoids -algoritmia hidastavaksi osaksi nousi klusterista uuden medoidin etsivä funktio, jossa algoritmi joutuu hakemaan jokaiselle klusterin alkiolle etäisyyden kaikkiin muihin klusterin alkioihin (ks. algoritmi 3). Etäisyysmatriisin esilaskennalla on verrattain pieni vaikutus k-medoids -klusterointialgoritmin ajoaikaan. Iris-aineistolla suoritetuissa testeissä alkuperäinen k-means -klusterointialgoritmi on hieman nopeampi, mutta generoidussa aineistossa, jossa on huomattavasti enemmän alkioita, alkuperäinen k-means -klusterointialgoritmi vaatii suuremman määrän iteraatioita konvergenssin saavuttamiseksi ja on tästä syystä hitaampi kuin k-means++.

Algoritmi	km. ajoaika (ms)	km. iteraatioiden lukumäärä (i)
K-means (Forgy)	205	5.87
K-means++	229	5.35
K-medoids	436	3
K-medoids (ennalta laskettu etäisyysmatriisi)	434	3

Taulukko 2. Algoritmien ajoaikojen tulokset Iris-aineistolla

5.3 Algoritmien tuottamien klustereiden tarkkuus

Algoritmien tuottamien klustereiden tarkkuuden vertailua varten generoitu aineisto sisälsi 800 alkiota, jokaisella alkiolla oli kolme piirrettä ja luokkamuuttuja, klustereiden lukumäärä oli kuusi ja klustereiden keskihajonta oli 1.25 (ks. kuva 6). K-means ja k-means++ -klusterointialgoritmit suoritettiin 10 kertaa, jonka jälkeen algoritmeille laskettiin keskimääräinen klustereiden luokittelutarkkuus (prosentuaalinen määrä oikein luokiteltuja alkioita) ottamalla luokittelutarkkuuden keskiarvo näistä kymmenestä suorituskerrasta. K-medoids -klusterointialgoritmi suoritettiin vain kerran, koska algoritmin medoidien aloitussijainnit ovat aina identtiset ja näin ollen algoritmi etenee aina samalla tavalla.



Kuva 6. Algoritmien oikeellisuuden vertailussa käytetty generoitu aineisto

Alkuperäinen k-means -klusterointialgoritmi tuotti parhaita tuloksia generoidulla aineistolla (ks. taulukko 3). Iris-aineistoin klusteroimisessa k-medoids

-klusterointialgoritmin tulokset ovat kuitenkin huomattavasti parempia kuin generoidulla aineistolla (ks. taulukko 4). Iris-aineistossa k-medoids -algoritmin ajoaika on lähes kaksi kertaa hitaampi kuin k-means++ -algoritmin (ks. taulukko 2), joka kuitenkin tuotti lähes vastaavia tuloksia.

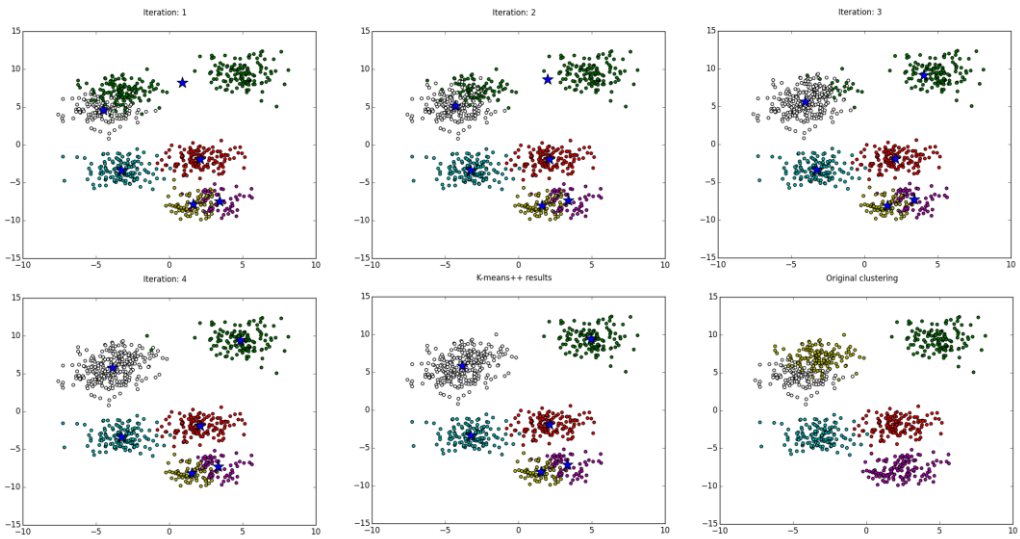
Algoritmi	km. luokittelutarkkuus (10 suor.)
K-means (Forgy)	83,61 %
K-means++	77,26 %
K-medoids	57,25 %

Taulukko 3. Algoritmien luokittelutarkkuuden vertailu, generoitu aineisto

K-means++ -klusterointialgoritmi tuotti keskimääräisesti heikkoja tuloksia generoidulla aineistolla (ks. kuva 7). K-medoidsin tuottamat klusterit generoidusta aineistosta olivat erittäin heikkoja verrattuna k-means -algoritmin tuloksiin. Saatuja tuloksia voi vääristää rajattu määrä testiajoja, mutta k-means++ ja k-medoids -algoritmien tuottamien heikkojen tulosten syynä oli todennäköisimmin sopimattomuus generoidun aineiston kanssa.

Algoritmi	km. luokittelutarkkuus (10 suor.)
K-means (Forgy)	81,99 %
K-means++	85,87 %
K-medoids	90,67 %

Taulukko 4. Algoritmien luokittelutarkkuus, Iris-aineisto



Kuva 7. K-means++ -algoritmin epäonnistunut klusterointi, generoitu aineisto

6. Yhteenveto

Variaatioita k-means -klusterointialgoritmista on kehitetty vuosien aikana huomattavia määriä, mutta alkuperäinen k-means -klusterointialgoritmi on kuitenkin käytetyin vielä nykypäivänä [Jain 2010]. K-means -klusterointialgoritmin suosio johtuu pitkälti sen yksinkertaisuudesta ja tehokkuudesta, ongelmana uusien varianttien käytössä on myös niiden implementaatioiden puute monista tutkijoiden ja ohjelmoijien hyödyntämistä tiedonlouhinta- ja koneoppimisalgoritmikirjastoista. Esitellyistä variaatioista suosituin oli selvästi k-means++ -klusterointialgoritmi, jonka toteutus löytyi useasta eri koneoppimisalgoritmikirjastosta.

Kokeellisissa vertailuissa k-means++ -algoritmi tuotti verrattain hyviä tuloksia, uhraamatta liikaa alkuperäisen k-means -klusterointialgoritmin laskennallista nopeutta. K-means++ -klusterointialgoritmi vaikuttaisi olevan lähes kaikilla tavoilla parempi kuin alkuperäinen k-means -algoritmi. K-medoids -algoritmin suurimmaksi ongelmaksi nousi sen hitaus, verrattuna k-means ja k-means++ -klusterointialgoritmeihin, mutta sen tuottamat klusterit Iris-aineistosta olivat lähimpänä totuutta kolmesta vertaillusta algoritmista.

Alkuperäinen k-means -klusterointialgoritmi on vielä nykypäivänäkin tarpeeksi nopea ja tuottaa verrattain hyviä tuloksia suurimpaan osaan käyttötarkoituksista. Klusterointialgoritmit ovat kuitenkin erittäin riippuvaisia aineiston laadusta, ja näin ollen mikään yksi klusterointialgoritmi ei selvästi tuota parhaita tuloksia riippumatta aineiston laadusta ja oikean klusterointialgoritmin valitseminen aineistolle on tärkein osa onnistunutta klusterointia.

Viiteluettelo

David Arthur and Sergei Vassilvitskii. 2006. How slow is the k-means Method?. In: *Proceedings of the Twenty-Second Annual Symposium on Computational Geometry (ACM)*, 144–153.

David Arthur and Sergei Vassilvitskii. 2007. k-means++: the advantages of careful seeding. In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07)*, 1027–1035.

Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar and Sergei Vassilvitskii. 2012. Scalable k-means++. In: *Proceedings of the Very Large Data Bases Endowment*, 5, 7, 622–633.

James C. Bezdek, Robert Ehrlich, and William Full. 1984. FCM: The Fuzzy C-means Clustering Algorithm. *Computers & Geosciences*, 10, 2, 191–203

P. S. Bradley, O. L. Mangasarian, and W. N. Street. 1997. Clustering via Concave Minimization. In: *Advances in Neural Information Processing Systems*, 368–374.

Adam Coates and Ng. Y. Andrew. 2012. Learning Feature Representations with K-means, *Neural Networks: Tricks of the Trade*. Springer.

G. Hamerly and C. Elkan. 2002. Alternatives to the k-means algorithm that find better clusterings. In: *Proceedings of the Eleventh International Conference on Information and Knowledge Management (ACM)*, 600–607.

J. A. Hartigan and M. A. Wong. 1979. Algorithm AS 136: A K-Means Clustering Algorithm. *Journal of the Statistical Society, Series C (Applied Statistics)*, 28, 1, 100–108.

J.D. Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, 9, 3, 90–95.

Anil K. Jain. 2010. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*. 31, 8, 651–666.

L. Kaufman and P.J. Rousseeuw. 1987. Clustering by means of Medoids. In: *Statistical Data Analysis Based on the L_1–Norm and Related Methods*, 405–416.

Kyoung-jae Kim and Hyunchul Ahn. 2008. A recommender system using GA K-means clustering in an online shopping market. In: *Expert Systems with Applications*, 34, 2, 1200–1209.

M. Lichman. 2013. UCI Machine Learning Repository: Iris-dataset. https://archive.ics.uci.edu/ml/datasets/Iris. Checked 16.12.2015.

C. D. Manning, P. Raghavan, and H. Schtze. 2009. *Introduction to Information Retrieval*. Cambridge University Press.

Marina Meilă. 2006. The Uniqueness of a Good Optimum for K-Means. In: *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, 625–632.

H.P. Ng, S.H. Ong, K. W. C. Foong, P. S. Goh, and W. L. Nowiski. 2006. Medical Image Segmentation using K-means Clustering and Improved Watershed Algorithm. In: *Proceedings of the Southwest symposium on Image Analysis and Interpretation, IEEE*, 61–65.

Hae-Sang Park and Chi-Hyuck Jun. 2009. A simple and fast algorithm for K-medoids clustering. *Expert Systems with Applications*, 36, 2, 2, 3336–3341.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michael, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 12, 2825–2830.

D. Pelleg and A.W. Moore. 2000. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In: *Proc. of the 17th International Conference on Machine Learning (ICML)*, 727–734.

Girish Punj and David W. Stewart. 1983. Cluster analysis in marketing research: review and suggestions for application. *Journal of Marketing Research*, 20, 2, 134-148.

S. Ray and Rose H. Turi. 1999. Determination of number of clusters in k-means clustering and application in colour image segmentation. In: *Proceedings of the 4th International Conference on Advances in Pattern Recognition and Digital Techniques*, 137–143.

E. Schubert, A. Koos, T. Emrich, A. Züfle, K. A. Schmid and A. Zimek. 2015. ELKI: Environment for Developing KDD-Applications Supported by Index-Structures, Example Data sets for ELKI: Mouse. http://elki.dbs.ifi.lmu.de/wiki/DataSets. Checked 16.12.2015.

Correct-by-construction-menetelmät ohjelmistotuotannossa

Mikko Paukkonen

Tiivistelmä.
Correct-by-construction-menetelmät ovat formaaleihin ohjelmistokehitysmenetelmiin kuuluvia menetelmiä, jotka keskittyvät ohjelman oikeellisuuteen itse toteutusmenetelmän oikeellisuudensäilyttävyyden kautta. Vaatimusmäärittelyn ollessa oikeellinen myös sen pohjalta johdettu ohjelma on oikeellinen. Tutkielmassa luodaan katsaus yleisellä tasolla formaaleihin menetelmiin ohjelmistotuotannossa ja erityisemmin correct-by-construction-menetelmiin formaalien menetelmien joukossa. Correct-by-construction helpottaa lähestymistapana joitakin muiden lähestymistapojen ongelmia.

Avainsanat ja -sanonnat: formaalit menetelmät, ohjelmistotuotanto, correct-by-construction, ohjelmien johtaminen.

1. Johdanto

Eräs ohjelmiston ominaisuus on sen oikeellisuus. Oikeellinen ohjelma toimii sille asetettujen vaatimusten mukaisesti. Tyypillisesti vaatimukset ovat kuitenkin itse ohjelmistotuotantoprosessissa esille tuodut vaatimukset. Ohjelma voi siis olla oikeellinen, mutta ohjelmistotuotannossa esille tuodut vaatimukset eivät kuitenkaan todellisuudessa vastaa esimerkiksi asiakkaan ohjelmistolle reaalikäytössä asettamia vaatimuksia.

Ohjelmistotuotannossa ohjelmiston oikeellisuus tyypillisesti todetaan testaamalla, katselmoimalla ja käyttäen erilaisia analysointityökaluja [Sommerville, 2007, 517]. Testaamisella ei kuitenkaan voida todistaa, ettei ohjelmistossa ole virheitä, vaan testaus ainoastaan voi paljastaa ohjelmistossa olevia virheitä. Virheettömyyden todistamiseen tarvitaan matemaattisesti täsmällisiä lähestymistapoja [Sommerville, 2007, 531].

Ohjelmoinnissa käytettävät formaalit menetelmät ovat joukko ohjelmistokehitykseen käytettäviä matemaattisia työkaluja. Formaaleille menetelmille ominaista on se, että ohjelma eri sen elinkaaren vaiheissa mallinnetaan matemaattisesti, minkä johdosta malleja voidaan analysoida ja varmentaa matemaattisen täsmällisesti [Woodcock et al., 2009], ja joissakin formaaleissa lähestymistavoissa on jopa mahdollista todistaa ohjelmiston virheettömyys [Abrial, 2006]. Kaikki formaalit menetelmät eivät kuitenkaan itsessään takaa ohjelmien oikeellisuutta, vaikka ne menetelminä ovatkin käyttökelpoisia ohjelmien mahdollisesti sisältämien virheiden löytämisessä [Clarke and Wing, 1996].

Tässä tutkielmassa luodaan katsaus correct-by-construction-menetelmiin, jotka ovat eräs formaalien menetelmien osa-alue. Correct-by-construction-menetelmät lähestyvät ohjelmiston oikeellisuutta sen toteuttamiseen käytettyjen menetelmien oikeellisuudensäilyttävyyden kautta: jos ohjelmiston vaatimusmäärittely vastaa sille asetettuja reaalimaailman vaatimuksia, myös suoritettava ohjelma on oikeellinen sen tuottamiseen käytetyn toteutusmenetelmän ominaisuuksista johtuen [Chapman, 2006].

Luvussa 2 esittelen formaalit menetelmät yleisesti. Luvussa 3 asetetaan correct-by-construction-menetelmät formaalien menetelmien viitekehykseen. Luvussa 4 annetaan esimerkki GCL-kielisen ohjelman johtamisesta määritelmän ja johtamissääntöjen avulla. Luku 5 omistetaan esiteltyjen asioiden yhteenvetoon.

2. Formaaleista menetelmistä

Ennen correct-by-construction-menetelmien esittelemistä on mielestäni aiheellista käsitellä formaaleja menetelmiä yleisesti. Woodcock ja muut [2009] määrittelevät formaalit menetelmät (formal methods) ohjelmistojen kehitystä varten käytettäviksi matemaattisiksi tekniikoiksi, joita voidaan käyttää ohjelmistojen analysointiin ja tarkastamiseen ohjelmiston kehitystyö jokaisessa vaiheessa.

Woodcock ja muut [2009] jakavat formaalien menetelmien käyttöalueita kuvatessaan ohjelmiston kehitystyön vielä hienojakoisempiin alueisiin: vaatimusmäärittelyyn, määrittelyyn, arkkitehtuuriin, suunnitteluun, toteutukseen, testaamiseen, ylläpitoon ja jatkokehitykseen. Sommerville [2007, 8] listaa ohjelmiston kehityskaaren vaiheiksi määrittelyn, toteutuksen, varmennuksen ja jatkokehityksen. Näiden vaiheiden välille on kuitenkin muodostettavissa vastaavuudet, joten tässä tutkielmassa ei aiheen käsittelyn yksinkertaistamisen vuoksi käsitellä ohjelmiston kehityskaaren vaiheita yhtä yksityiskohtaisesti kuin Woodcock ja muut [2009], vaan tutkielmassa pitäydytään Sommervillen neliosaisessa jaottelussa, joka on esitetty kuvassa 1.

Sommerville [2007]	Woodcock ja muut [2009]
vaatimusmäärittely [s. 75]	vaatimusmäärittely
toteutus [ss. 76 - 80]	määrittely
	arkkitehtuuri
	suunnittelu
	toteutus
varmennus [ss. 80 - 81]	testaaminen
jatkokehitys [ss. 81 - 82]	ylläpito
	jatkokehitys

Kuva 1. Ohjelmiston kehityskaaren vaiheet Sommervillen [2007] sekä Woodcockin ja muiden [2009] mukaan.

Kehityskaaren ajattelu nelivaiheisena ei kuitenkaan tarkoita, että luetellut vaiheet seuraisivat koko ohjelmiston kehitysajan toisiaan, vaan nämä vaiheet voivat eri kehitysmallissa seurata toisiaan tai limittyä keskenään käytettävästä ohjelmistoprosessimallista (software process model) riippuen vaihtelevissa määrin [Sommerville, 2007, 65]. Tämän tutkielman käsitellessä eri osa-alueita onkin hyvä pitää mielessä osa-alueiden tulosten luonteen tärkeys verrattuna niiden ajalliseen sijoittumiseen ohjelmiston kehityskaarella.

2.1. Vaatimusmäärittelyssä

Ohjelmiston määrittelyllä (requirements engineering) tarkoitetaan järjestelmän ja siltä haluttujen ominaisuuksien, vaatimusten, määrittelemistä [Clarke and Wing, 1996].

Vaatimuksia kuvataan ei-formaalissa ohjelmistokehityksessä luonnollisella kielellä tai kaavakuvin. Esimerkkejä kaavakuvista ovat esimerkiksi sekvenssidiagrammit ja käyttötapauskuvaukset [Sommerville, 2007, 131]. Luonnollisen kielen monitulkintaisuuden vuoksi luonnollisella kielellä kirjoitetussa vaatimusmäärittelyssä kielenkäyttö on strukturoidumpaa kuin puhtaan luonnollinen teksti [Sommerville, 2007, 130–131].

Formaalissa vaatimusmäärittelyssä kuvaus tapahtuu rakenteilla, joiden muoto ja merkitys on määritetty matemaattisesti [Clarke and Wing, 1996]. Vaatimukset voivat liittyä järjestelmän käyttäytymiseen, ajoitukseen, suorituskykyyn tai ohjelman sisäiseen rakenteeseen. Clarken ja Wingin [1996] mukaan tähän mennessä formaaleita menetelmiä käyttämällä on suurinta menestystä saavutettu järjestelmän käyttäytymisen määrittelyssä.

Kuten ei-formaalissa vaatimusmäärittelyssä [Sommerville, 2007, 75–76], myös formaalissa vaatimusmäärittelyssä saavutetaan tehdyn työn tuloksena yksityiskohtaisempaa ymmärrystä ohjelmistolta vaaditusta toiminnasta [Sommerville, 2007, 221]. Formaalin vaatimusmäärittelyn tuloksena on kuitenkin myös

matemaattisesti analysoitava vaatimusjoukko. Esimerkiksi vaatimusten sisäinen johdonmukaisuus voidaan tarkistaa mekaanisesti [Clarke and Wing, 1996].

2.2. Toteuttamisessa

Ohjelmiston toteutus tarkoittaa suorituskelpoisen järjestelmän muodostamista ohjelmiston vaatimusmäärittelyn pohjalta [Sommerville, 2007, 76]. Toteuttamisen aikana suunnitellaan esimerkiksi ohjelmiston arkkitehtuuria, järjestelmän eri osien välisiä rajapintoja ja ohjelmistossa käytettäviä algoritmeja ja tietorakenteita [Sommerville, 2007, 77]. Sommerville [2007, 77] listaa suunnittelun vaiheiksi arkktehtuurin, abstraktin määrittelyn, rajapintasuunnittelun, komponenttisuunnittelun, tietorakenteiden ja algoritmien suunnittelun. Sommervillen [2007, 78] mukaan kaksi viimeistä voidaan jättää varsinaiseen toteutusvaiheeseen.

Ei-formaaleissa menetelmissä arkkitehtuurin ja järjestelmän tarkemman toiminnallisuuden toteuttamisessa voidaan käyttää apuvälineenä erilaisia tilakaavioita, eri ohjelman olioiden vuorovaikutusta kuvaavia kaavioita ja sekvenssikaavioita [Sommerville, 2007, 78]. Varsinainen ohjelmointi on Sommervillen [2007, 79] mukaan varsin yksilöllinen prosessi, jossa ei noudateta juuri yleistä yhtenäistä prosessia. Esimerkiksi jotkin ohjelmoijat toteuttavat tutunoloiset komponentit ensin ja toiset taas tekevät vieraanoloiset komponentit ensin. Jotkut saattavat suunnitella tietorakenteet valmiiksi jo alkuvaiheessa, jolloin ne ohjaavat muuta kehittämistä kun taas muut jättävät niiden määrittelyn myöhäisempään vaiheeseen. Debuggaaminen on eräs prosessi, jolla korjataan ohjelmavirheitä. Prosessissa virheen paikantaminen, korjauksen suunnittelu ja toteuttaminen ja ohjelman uudelleenkokeilu seuraavat järjestyksessä toisiaan [Sommerville, 2007, 79]. Kehitysvaiheessa tehdään siis yleensä myös jonkinlaista testaamista, jolloin se on osa niin ohjelman varmentamista kuin toteuttamistakin [Sommerville, 2007, 79].

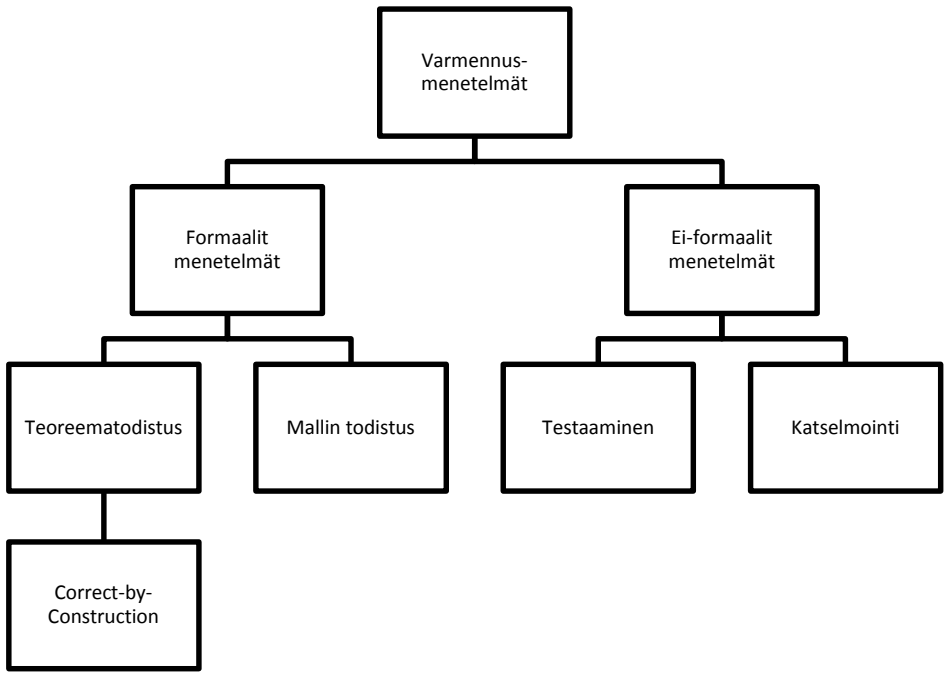
Eräs tapa, jolla formaaleja menetelmiä integroidaan muutoin ei-formaaliin ohjelmistokehitysprosessiin, ovat niin kutsutut kevyet formaalit menetelmät (lightweight formal methods). Kevyitä lähestymistapoja ovat esimerkiksi staattisen analyysin käyttö tai formaalin määrittelyn tulosten siirtäminen muulla menetelmällä kehitettyyn ohjelmaan esimerkiksi ajonaikaisiksi esi- ja jälkiehdoiksi (assertions) [Woodcock et al., 2009].

Formaalimpi tapa tehdä itse kehitystä on käyttää ohjelmointikieltä, joka tukee formaaleja ohjelmistokehitysmenetelmiä. On olemassa kieliä, jotka on johdettu muista kielistä lisäämällä niihin annotaatioita, joiden avulla ohjelmaa voidaan analysoida tarkemmin [Chapman, 2006], kun taas jotkin kielet on alusta alkaen tehty formaalia ohjelmistokehitystä silmällä pitäen [Dijkstra, 1975]. For-

maalia sovelluskehitystä voi tehdä myös hyödyntämällä itse vaatimusmäärittelyä varsinaisen toteutuksen pohjana ja pyrkiä soveltuvia työkaluja käyttäen johtamaan toteutus vaatimusmäärittelyn pohjalta. Tätä lähestymistapaa kutsutaan johtamispohjaiseksi ohjelmoinniksi (derivational programming) [Smith and Hoebel, 2010].

2.3. Varmentamisessa

Ehkä tärkein formaalien menetelmien käyttötarkoitus ja ero ei-formaaleihin menetelmiin on niiden käyttömahdollisuudet ohjelman toiminnallisuuden varmentamisessa. Varmentaminen tarkoittaa ohjelmiston vaatimusten ja toteutetun toiminnallisuuden vertaamista niiden toisiaan vastaavuuden toteamiseksi. Oikeellinen ohjelmisto toteuttaa sille asetetut vaatimukset [Sommerville, 2007, 80]. Kuvassa 2 on esitetty eräänlainen jaottelu esiteltyjen varmennusmenetelmien välillä.



Kuva 2. Eräiden ohjelmiston varmennusmenetelmien jaottelu.

Ei-formaaleissa menetelmissä testaaminen, joka voi olla automatisoitua, on suurityöinen osa ohjelmistokehitystä. [Sommerville, 2007, 561]. Testaamisella ei kuitenkaan voida varmistua ohjelmiston virheettömyydestä, sillä testaaminen ei voi todistaa virheettömyyttä, vaan ainoastaan paljastaa virheiden olemassaolon. Testaaminen on kuitenkin tapa hankkia luottamusta siihen, että ohjelmisto toimii oikeellisesti [Sommerville, 2007, 539].

Katselmointi on testaamisen rinnalla niin kutsuttu staattinen menetelmä. Ohjelmaa ei tarvitse ajaa sen oikeellisuuden toteamiseksi, vaan sen oikeellisuus pyritään toteamaan sen toteutusta tarkastelemalla esimerkiksi automatisoiduilla

analyysityökaluilla tai katselmoijana toimivan ohjelmoijan toimesta [Sommerville, 2007, 517].

Formaaleihin menetelmiin kuuluu menetelmiä, joilla ohjelmiston oikeellisuus voidaan todistaa matemaattisesti. Woodcock ja muut [2009] kuvaavat vaatimuksien ja ohjelmien suhdetta siten, että jokaista vaatimusmäärittely-ohjelma -paria vastaa oikeellisuusteoreema. Teoreeman olemassaolo yritetään todistaa. Jos tällainen oikeellisuusteoreema on olemassa, ohjelma on oikeellinen; ohjelma tekee juuri sen, mitä vaatimusmäärittely sille vaatimuksiksi asettaa.

Tällaisen teoreeman muodostamiseen on laajassa käytössä kaksi vaihtoehtoista lähestymistapaa: teoreemapohjainen ja mallipohjainen. Vaikka ensin mainitun nimi onkin helposti sekoittuva, tarkoitetaan sillä lähestymistapaa, jossa ohjelmisto ja sen ominaisuudet on ilmaistu matemaattista logiikkaa hyväksikäyttäen, kun taas jälkimmäisenä mainitussa mallipohjaisessa lähestymistavassa ohjelmistosta muodostetaan äärellinen malli, jonka tilat käydään läpi [Clarke and Wing, 1996].

On kuitenkin huomattava, että oikeaksi todistamisen hyödyllisyys perustuu tärkeään seikkaan: pohjana olevien reaalimaailman vaatimusten ja vaatimusten, joiden pohjalta ohjelma on tehty, tulee vastata toisiaan [Sommerville, 2007, 531]. Oikeellisuustodistus ei siis ole välttämättä tae siitä, että ohjelma tekee, mitä sen halutaan tekevän.

2.4. Jatkokehityksessä

Jatkokehitys tarkoittaa ohjelmiston kehitysvaiheita sen varsinaisen vaatimusmäärittelyn, toteuttamisen ja varmentamisen jälkeen, kun ohjelmisto on käytössä [Sommerville, 2007, 489]. Vaikka vaiheen voikin ajatella olevan erillinen, se pitää sisällään tavallaan kaikki edellä mainitut vaiheet [Sommerville, 2007, 489]: yleinen syy ohjelmiston jatkokehitykseen on ohjelmiston toiminnalle asetettujen vaatimusten muuttuminen. Vaatimusten muuttumista seuraa vaatimusmäärittely, uusien vaatimusten toteuttaminen ja ohjelmiston toiminnan varmentaminen.

Vaikka nämä vaiheet ovatkin tavallaan ohjelmiston alkuperäisen kehityksen jatkumoa, on formaalien ja ei-formaalien menetelmien käyttöä verratessa mielenkiintoista tarkastella tilannetta, jossa uuden ohjelmiston sijaan jatkokehitetään olemassa olevaa ohjelmistoa, sillä ohjelmistotyön piirteet ohjelmiston jatkokehitystilanteessa eroavat jonkin verran niiden käytöstä uutta sovellusta kehitettäessä, sillä jatkokehityksessä työmäärää lisää ohjelmiston ymmärtämiseen tarvittava työ [Sommerville, 2007, 489].

Ei-formaaleissa menetelmissä jatkokehityksen tehokkuus riippuu suuresti siitä, miten hyviä toteutusmenetelmiä niiden toteuttamisessa on käytetty ja miten tehdyt ratkaisut on dokumentoitu ja miten hyvin rakenteita ylläpidetään ja korjataan jatkokehityksen aikana [Sommerville, 2007, 495].

Formaaleja menetelmiä käytettäessä muutosten tekeminen voi olla jossain määrin ei-formaaleja menetelmiä tehokkaampaa niin ajallisesti, rahallisilta kustannuksiltaan kuin tehtyjen virheiden määrällä mitattuna. Formaalien menetelmien edut ohjelmiston oikeellisuuden varmennuksessa johtavat siihen, että vaatimusten muutoksista huolimatta on helpompaa todeta ohjelman toteuttavan sille asetetut uudet vaatimukset samalla, kun se edelleen toteuttaa myös aiemmat vaatimukset virheettä [Woodcock et al., 2009].

2.5. Formaalien menetelmien käyttö ohjelmistotuotannossa

Vaikka formaaleilla menetelmillä katsotaankin olevan hyödyllisiä ominaisuuksia virheettömyyden kannalta ohjelmistokehityksen eri vaiheissa ja tutkijoiden ja menetelmien käyttäjien keskuudessa on pidetty menetelmien enenevää käyttöä ohjelmistotuotannossa tavoiteltavana, ei formaalien menetelmien käyttö nykyaikaisessa ohjelmistotyössä ole erityisen laajaa lukuun ottamatta tiettyjen alojen kriittisten järjestelmien kehittämistä [Woodcock et al., 2009].

Syynä pidetään esimerkiksi näytön vähyyttä, koska menetelmien kustannustehokkaasta käytöstä ei katsota olevan tarpeeksi empiirisiä todisteita [Woodcock et al., 2009]. Tätä ongelmaa ratkaistakseen Woodcock ja muut [2009] ovat koonneet yhteen todisteiksi joukon erilaisia projekteja, joissa formaaleja menetelmiä on käytetty. Lisäksi menetelmien hyödyntämisessä käytettävien työkalujen käytettävyydessä on ongelmia, vaikka kehitys kulkeekin helpommin käytettäviä työkaluja kohti, esimerkiksi matemaattisen notaation ja graafisten esitysten yhdistäminen on yleistyvää samoin kuin olemassa oleviin kehitysprosesseihin integroituvat menetelmät. [Woodcock et al., 2009].

3. Correct-by-construction-menetelmät formaalien menetelmien joukossa

Correct-by-construction-menetelmät ovat yksi keino tehdä ohjelmistokehitystä formaalein menetelmin. Tässä luvussa käsittelen correct-by-construction-menetelmiin kuuluvia tekniikoita, joita ohjelmistotyön eri vaiheissa voidaan käyttää, ja vertaan niitä joihinkin muihin formaaleihin ohjelmistokehitysmenetelmiin.

3.1. Toteutuksessa

Correct-by-construction-menetelmissä itse ohjelmien toteutus perustuu pitkälti niin sanottuun laskennalliseen ohjelmointityyliin (calculational style) [Chaudhari and Damani, 2015]. Siinä on tärkeää, että ohjelman muodostaminen aloitetaan vaatimusmäärittelyn tuloksista. Vaatimusmäärittelyn tuloksista muodostetaan formaali määrittely. Formaali määrittely on ilmaistu esi- ja jälkiehtoina. Ohjelma muodostetaan johtamalla ohjelma formaalista määrittelystä. Johtaminen tapahtuu soveltamalla oikeellisuuden todistettavasti säilyttäviä johtamissääntöjä [Smith and Hoebel, 2010; Kourie and Watson, 2012, 7].

Jokaisessa johtamisen vaiheessa valitaan sääntö, jonka ajatellaan johtavan lähemmäksi lopullista suorituskelpoista ohjelmaa. Johtamissäännön valinta ei aina ole kuitenkaan itsestäänselvää, joten oikeanlaisen johtamisketjun muodostaminen vaatii kuitenkin toteuttajalta jonkinasteista luovuutta [Smith and Hoebel, 2010].

Eräs lähestymistavan etu on se, että on helpompaa huomata missä kohtaa ja miten virhe on tehty kuin jos ohjelma ensin toteutettaisiin ja sen jälkeen varmennettaisiin [Chaudhari and Damani, 2015]. Lisäksi ohjelman johtamisketju toimii ohjelman dokumentaationa, kun johtamisaskeleet dokumentoivat tehdyt suunnitteluvalinnat [Smith and Hoebel, 2010].

3.2. Varmentamisessa

Correct-by-construction-menetelmissä olennainen piirre ohjelmiston oikeellisuuden todistamisen kannalta on se, että ohjelman oikeellisuutta lähestytään itse rakennusmenetelmän oikeellisuuden kannalta. Jos ohjelmiston vaatimusmäärittely vastaa todellisia vaatimuksia ja johtamissääntöjä on noudatettu oikein, myös lopullinen suorituskelpoinen ohjelma on oikeellinen [Chaudhari and Damani, 2015].

Oikeellisuuden lisäksi correct-by-construction-menetelmien eräs etu on se, että ohjelman johtamiseen käytetty johtamisvaihdeiden ketju antaa vihjeitä siitä, missä kohti ohjelmassa mahdollisesti oleva virhe on [Chaudhari and Damani, 2015].

Toinen merkittävä lähestymistapa on todeta ohjelman oikeellisuus jälkikäteen muodostamalla todistus ohjelman oikeellisuudesta. Yksi iso tämän lähestymistavan puute on se, että monimutkaisen ohjelman oikeellisuuden todistaminen on hyvin vaikea tehtävä, jos todistus joudutaan tekemään kokonaisuudessaan jälkikäteen [Chaudhari and Damani, 2015].

Correct-by-construction-menetelmissä voidaan myös käyttää johdetun ohjelman dokumentoimiseen sen johtamisketjua [Smith and Hoebel, 2010]. Johtamis-

ketju muodostuu niistä johtamissäännöistä, joilla formaalista vaatimusmäärittelystä on päädytty suoritettavaan ohjelmaan. Näin ohjelman johtamisketju perusteluineen muodostaa dokumentaation siitä, miten oikeelliseen ohjelmaan on päädytty.

3.3. Jatkokehityksessä

Johtamalla tehtyjen ohjelmien ominaisuus on se, että ohjelma tavallaan on sen johtamiseen käytetty ketju. Varsinaista ohjelmakoodia ei ohjelmoija muokkaa, vaan muutokset ohjelmaan tehdään sen johtamisketjua muokkaamalla. Tästä on etuja. Ohjelman jatkokehityksessä olemassa olevaan johtamisrakennetta voidaan hyödyntää automatisoidussa ohjelmanmuodostuksessa [Smith and Hoebel, 2010].

Ohjelman muutokset muodostuvat jopa automatisoidusti muuttuvien vaatimusten ja johtamisrakenteen muutoksen seurauksena. Vaikka johtamisketjua jouduttaisiin riittävän suurten vaatimusmuutosten seurauksena muuttamaan, voidaan aiempaa johtamisketjua käyttää hyödyksi uutta johtamisketjua kehittäessä [Smith and Hoebel, 2010].

Johtamiskeskeinen ohjelmistokehitys voi jopa siirtää ohjelmistokehityksen työnä enemmän metatasolle, kun ohjelmoijat ohjelmakoodin sijasta tekevät suunnitteluratkaisuja järjestelmässä [Smith and Hoebel, 2010], jossa ei välttämättä tarvitse tuntea itse johtamisrakenteita niin syvällisesti. Tietenkin on edelleen oltava niitä, jotka nämä rakenteet tuntevat.

4. Perättäishaun toteutuksen johtaminen

Kuten luvussa 3 mainittiin, ohjelman johtaminen tarkoittaa ohjelman muodostamista aluksi sen formaalista määrittelystä vaihe vaiheelta monimutkaisemmista ohjelmista lopulta päädyttäessä ohjelmaan, joka on määritelty riittävän tarkasti sen ollakseen suoritettavissa.

4.1. Guarded Command Language

GCL on Dijkstran [1975] kehittämä ohjelmointikieli. Kielelle ominaista on sen valinta- ja toistorakenteiden muodostuminen joukosta vahdittuja (guarded) komentoja. Komento koostuu komentojonosta ja sitä edeltävästä predikaatista, jonka on oltava tosi, jotta komentojono voidaan ottaa suoritettavaksi. Tämä ominaisuus mahdollistaa epädeterministisen suorituksen, jossa ohjelman suorituspolkua tai välttämättä edes sen jälkeistä tilaa ei voi päätellä ohjelman alkutilasta. Dijkstran [1975] mukaan tämä helpottaa ohjelmien järjestelmällistä johtamista. Tässä luvussa käytetään ohjelman johtamisesta annetun esimerkin esittämisessä GCL-kieltä.

4.2. Ohjelmien ilmaiseminen

Ohjelmien tarkastelemiseksi on syytä esittää muutamia merkintätapoja ja käsitellä tarkemmin, mikä ohjelma tarkalleen ottaen on. Hoare [1969] esittää ohjelmat ja siihen liittyvät esi- ja jälkiehdot merkinnällä $P\{Q\}R$, missä P on ohjelman esiehdot, Q ohjelma ja R ohjelman jälkiehdot.

Ehdot muodostuvat predikaatista, jonka totuusarvo riippuu järjestelmän tilasta. Ennakkoehtojen tulee olla tosia, jotta ohjelman suoritus on mahdollista ja jälkiehdot ovat ehdot, joiden tulee päteä ohjelman suorituksen jälkeen [Hoare 1969]. Toisin ilmaistuna järjestelmän tilan tulee olla jossain sellaisessa järjestelmän tilajoukon tilassa, jossa ohjelman ennakkoehdot ovat tosia ja järjestelmän tulee olla ohjelman jälkiehdot täyttävässä järjestelmän mahdollisten tilojen joukon tilassa, jotta ohjelma voidaan katsoa oikeellisesti suoritetuksi.

Puhtaan teoreettisessa tarkastelussa voidaan pitäytyä määrittelemään vain muutamia järjestelmän tilaa kuvaavia matemaattisia muuttujia, joihin predikaatin totuusarvo perustuu. Reaalimaailman sovelluksia mallinnettaessa voidaan koneen tila mallintaa esimerkiksi sen muistiavaruutta kuvaavana alkiojonona, jossa jokaista osoitetta vastaava sisältö on kuvattu [Kourie and Watson, 2012, 11].

Oikeellisuuden käsitettä käsiteltiin jo jossain määrin ei-formaalina käsitteenä aiemmin tässä tutkielmassa. Oikeellisuudelle on ohjelmien johtamisen kontekstissa kuitenkin tärkeää olla täsmällisempi merkitys. Hoare [1969] määrittelee em. notaation esittelyn yhteydessä myös ohjelman oikeellisuuden siten, että oikeellisuus on saavutettu, jos ohjelma sen ennakkoehtojen pätiessä päätyy suorituksensa johdosta annettuihin jälkiehtoihin.

4.3. Oikeellisuus ja totaalinen oikeellisuus

Hoare [1969] huomauttaa, että hänen esittelemänsä rakenteet eivät ole avuksi todistettaessa, että ohjelman suoritus päättyy onnistuneesti. Epäonnistunut suoritus voi Hoaren mukaan johtua ikuisesta silmukasta, kielen toteutuksen rajan rikkomisesta tai esimerkiksi käyttöjärjestelmän asettamasta aikarajasta ohjelman suoritukselle.

Dijkstra [1975] laajentaa Hoaren rakennetta esittelemällä heikoimman esiehdon (weakest precondition) käsitteen. Heikoin esiehto on sellainen esiehto, jonka pätiessä ohjelma voi päättyä pysähtymättä annettuun jälkiehtoon ja jota heikompia edellisen vaatimuksen täyttäviä esiehtoja ei ole olemassa. Se on siis heikoin mahdollinen esiehto, joka takaa ohjelman suorituksen päätymisen jälkiehdon täyttävään tilaan onnistuneesti.

Vaikka ohjelmointia harjoittanut henkilö voisikin ehkä helposti muodostaa jonkinlaisen intuitiivisen käsityksen siitä, miten ehtojen heikkoutta ja vahvuutta

verrataan keskenään, on aiheen formaalin käsittelyn kannalta hyödyllistä määritellä ehdon vahvuus tarkasti.

Tarkastellaan ehtoja A ja B. Ehto A on heikompi kuin ehto B, jos ehto B pätee aina, kun ehto A pätee. Predikaattilogiikassa $A \rightarrow B$ on tautologia. Tämä ei tietenkään sulje pois sitä, että ehtojen välillä pätisi ekvivalenssi, mutta se ei tässä tutkielmassa, eikä tässä asiayhteydessä, ole tärkeää. Tämä siksi, koska kiinnostavaa on ainoastaan se, onko jokin ehto heikompi ekvivalenssin ollessa täysin yhdentekevää heikomman esiehdon määrittelyn kannalta.

4.4. GCL-kielen rakenteiden semantiikka

GCL-kielinen ohjelma koostuu Kuva 3 esitetyistä rakenteista. Tässä kohdassa kerrotaan lyhyesti, mitkä ovat kielen eri rakenteisiin liittyvät heikoimmat esiehdot Kourien ja Watsonin [2012, 17–34] mukaan. Rakenteiden määrittelyn tarkempi analyysi on tässä tarkastelussa jätetty pois, mutta rakenteiden semantiikan ovat tarkemmin määritelleet esimerkiksi Kourie ja Watson [2012]. Rakenteen x heikoimmasta esiehdosta käytetään merkintää $wp(x, Q)$, missä Q on rakenteen jälkiehto. Eri rakenteiden heikoimpia on annettu tässä tarkastelussa lähinnä esimerkinomaisesti, eikä tässä kohdassa esitettävä tarkastelu siten ole täydellinen.

Rakenne	Merkintä
Tyhjä komento	$skip$
Sijoitus	$:=$
Kompositio	;
Valinta	**if**
Toisto	**do**

Kuva 3. GCL-kielen rakenteet.

Tyhjä komento ei muuta ohjelman tilaa mitenkään. Näin olleen tyhjän komennon heikoimman esiehdon on oltava sama kuin jälkiehto, siis $wp(x, Q) = Q$.

GCL-kielessä on määritelty niin yksittäisen kuin useammankin muuttujan yhtäaikainen sijoitus. Käsitellään yksittäisen muuttujan tapaus. Sijoitus vasemmalla puolella olevaan muuttujaan oikealla puolella olevan arvon. $wp(x := E, Q) = Q[x\backslash E]$. Merkintä \ tarkoittaa muuttujaan sijoittamista. Sijoituksen yleistäminen useamman muuttujan yhtäaikaiseen sijoittamiseen on sellainen, että muuttujien arvot sijoitetaan yhtäaikaisesti, eikä sijoituksilla ole vaikutusta toisiinsa.

Kompositio on rakenne, jolla voidaan yhdistää kaksi komentoa yhdeksi rakenteeksi. Olkoon $S1$ ja $S2$ komennot. Tässä tilanteessa $wp(S1; S2, Q) = wp(S1, wp(S2, Q))$. Ohjelmia johdettaessa on kätevää huomioida se, että jos on olemassa sellainen predikaatti, joka toimii samanaikaisesti S1:n jälkiehtona ja

S2:n esiehtona, on kompositio voimassa. Jos $(\{P\}S1\{M\}) \wedge (\{M\}S2\{Q\})$ niin $\{P\}S1; S2\{Q\}$. Tämä ominaisuus helpottaa jälkiehtoon saapumista pienemmissä askelissa, kun voidaan ensin miettiä yhdistettyjen komentojen jälkiehtoon saapumista välivaiheen kautta.

GCL-kielen valintarakenne poikkeaa muista ohjelmointikielistä melkoisesti. Valintarakenne on muotoa:

if $G_1 \rightarrow S_1$
[] $G_2 \rightarrow S_2$
...
[] $G_n \rightarrow S_n$
fi,

missä G_i on predikaatti ja S_i on komento. G_i ja S_i muodostavat yhdessä vartioidun komennon (guarded command). Kaikista vartioiduista komennoista valitaan suoritettavaksi sellainen, jonka predikaatti on tosi. Jos vartioiduista komennoista useamman predikaatti on tosi, suoritettava komento valitaan mielivaltaisesti. Jos mikään predikaateista ei ole voimassa, komento päättää ohjelman suorituksen.

Toistorakenne on muotoa:

do $G_1 \rightarrow S_1$
[] $G_2 \rightarrow S_2$
...
[] $G_n \rightarrow S_n$
od.

Toistorakenteen heikoimman esiehdon määritelmä on verrattain monimutkainen, eikä sitä käsitellä kaikissa oppikirjoissakaan [Kourie and Watson, 2012, 33–34]. Kourie ja Watson [2012, 34] kuitenkin päätyvät tulokseen, että toistorakenne taatusti suoritetaan loppuun onnistuneesti, jos jokainen rakenteen komento voidaan suorittaa loppuun onnistuneesti.

4.5. Johtamissäännöt

Dijkstra [1975] esittelee kielen ja joukon johtamissääntöjä (derivation rules), joka on suunniteltu nimenomaan ohjelmien johtamista silmällä pitäen. Kieli laajentaa jossain määrin Hoaren [1969] esittelemää ajatusta, jossa ohjelma ja sen odotettu toimintatapa esitetään ohjelman ennakkoehtojen, jälkiehtojen ja itse ohjelman yhdistettynä kolmikkona.

Ohjelman johtaminen aloitetaan formaalista määrittelystä, josta sitten oikeiksi todistettuja muunnossääntöjä hyväksikäyttäen muodostetaan suoritettava ohjelma. Jotta ohjelmien johtaminen on mahdollista, joudutaan ensin määrittele-

mään joitakin määrittelyssä käytettyjä rakenteita ja muunnossäännöt, joiden perusteella ohjelmaa voidaan muokata. Tällaisia laskusääntöjä ovat esimerkiksi määritelleet Hoare [1969], Dijkstra [1975] ja Kourie ja Watson [2012] Dijkstran [1975] työtä apuna käyttäen. Kuva 4 on esitetty edellä mainitut johtamissäännöt. Sääntöjen tiiviimmän esitystavan vuoksi kuvassa on käytetty Morganin [1998] merkintätapaa. Merkintä $x \Rrightarrow y$ tarkoittaa "aina jos x, niin y". Merkintä $x \sqsubseteq y$ tarkoittaa "y on johdettu x:stä", ohjelmasta x voi siis johtaa ohjelma y:n. Sääntö on merkitty muodossa $w: [P, Q]$, missä w tarkoittaa kehysmuuttujia, P esiehtoja ja Q jälkiehtoja. Itse ohjelmaa merkintätavassa ei ole. Toistorakenteessa käytetty variantti on kokonaisluku, joka vähenee jokaisella toistokerralla ja jolla on jokin määritelty alaraja. Merkintä i_0 tarkoittaa muuttujan i arvoa ennen ohjelman osan suorittamista.

Sääntö no.	Nimi	Soveltamisehto ja sääntö
1	Jälkiehdon vahvennus	Jos $Q' \Rrightarrow Q$ niin $w: [P, Q] \sqsubseteq w: [P, Q']$
2	Esiehdon heikennys	Jos $P \Rrightarrow P'$ niin $w: [P, Q] \sqsubseteq w: [P', Q]$
3	Tyhjä komento	Jos $P \Rrightarrow Q$ niin $w: [P, Q] \sqsubseteq skip$
5	Sijoitus	Jos $P \Rrightarrow Q[\backslash E]$ niin $w: [P, Q] \sqsubseteq w \coloneqq E$
6	Sekventiaalinen kompositio	$w: [P, Q] \sqsubseteq w: [P, R]; w: [R, Q]$
7	Seuraava sijoitus	$w, x: [P, Q] \sqsubseteq w, x: [P, Q[x\backslash E]]; x \coloneqq E$
8	Valinta	Jos $P \Rrightarrow G_1 \vee G_2 \dots G_n$ niin $w: [P, Q] \sqsubseteq$ **if** $G_1 \rightarrow w: [G_1 \wedge P, Q]$ ... $[\!] \; G_n \rightarrow w: [G_n \wedge P, Q]$ **fi**
9	Toisto	Jos $GG = G_1 \wedge G_2 \dots G_n$ ja V on variantti, niin $w: [P, P \wedge \neg GG] \sqsubseteq$ **do** $G_1 \rightarrow w: [P \wedge G_1, P \wedge (0 \leq V < V_0)]$ ... $[\!] \; G_n \rightarrow w: [P \wedge G_n, P \wedge (0 \leq V < V_0)]$ **od**

Kuva 4. Ohjelman johtamissäännöt.

4.6. Perättäishaun johtaminen

Kourie ja Watson [2012, 56–60] käsittelevät perättäishaun toteutuksen johtamista. Ennen itse toteutuksen johtamista muodostetaan ohjelmalle sopivat esi- ja jälkiehdot. Tarkastellaan taulukkoa A ja muuttujaa x. Taulukon alkioiden ja muuttujan x tyyppi ovat samat. Oletetaan, että taulukossa A on alkio, jonka arvo on x. Ohjelman suorituksen loputtua on alkion A_i (taulukon indeksit ovat $0 \ldots A.len - 1$, missä $A.len$ on taulukon alkioiden määrä) oltava ohjelman suorituksen jälkeen sellainen, jonka arvo on A. Indeksi i osoittaa siis jonkin x-arvoisen alkion sijainnin taulukossa.

Esiehto on siis se, että taulukossa on alkio, jonka arvo on x. Jälkiehto on edellä esitetyn mukaisesti se, että i on x:n indeksi taulukossa. Nämä voidaan esittää Morganin [1998] notaatiossa muodossa $i{:}\,[appears(A, x, 0, A.len), (A_i = x)]$. $appears(A, x, i, j)$ on tosi jos ja vain jos taulukossa A esiintyy x välillä $[i, j[$ [Kourie and Watson, 2012, 56].

Haetaan alkiota peräkkäin oikealta vasemmalle. Muodostetaan silmukkainvariantti, joka pätee aina ennen toistoa ja sen jälkeen ja joka myös jotenkin esittää toivottua hakujärjestystä. Muodostetaan invariantti $inv = \neg appears(A, x, i + 1, A.len)$, joka kertoo, että x ei löydy jo haetusta osasta taulukkoa $[i, A.len[$. Vahvennetaan kuvan 4 säännön 1 mukaisesti ohjelman jälkiehtoa, jolloin ohjelma on $i{:}\,[appears(A, x, 0, A.len), (A_i = x) \wedge inv]$.

Jotta invariantti saataisiin voimaan, on se erikseen ohjelman suorituksen aikana tehtävä. Tätä varten on kätevää käyttää sekventiaalista kompositiota: $i{:}\,[appears(A, x, 0, A.len), inv]; i{:}\,[inv, inv \wedge (A_i = x)]$. Näin voidaan aluksi keksiä, miten alkuehdosta päästään invarianttiin ja miten invariantista päästään lopuksi tilanteeseen jossa invariantti ja alkuperäinen jälkiehto pätee [Kourie and Watson, 2012, 57].

Korvataan invariantti ensimmäisessä osassa sen määritelmällä: $i{:}\,[appears(A, x, 0, A.len), appears(A, x, i + 1, A.len)]$. Ei ole vaikeaa nähdä, että sijoitus $i \coloneqq A.len - 1$ on sopiva toimenpide jälkiehtoon pääsemiseksi. Alkuperäisessä esimerkissä [Kourie and Watson, 2012, 58] on esitetty tarkempi perustelu sijoituksen pätevyydestä. Ohjelma on tämän johtamisaskeleen jälkeen $i \coloneqq A.len - 1; i{:}\,[inv, inv \wedge (A_i = x)]$. Alkuosa on siis nyt yksinkertainen sijoitus.

Ohjelman jälkimmäisestä osasta on johdettavissa toistorakenne [Kourie and Watson, 2012, 59]. Konjunktion jälkiosasta $(A_i = x)$ voidaan johtaa kuvan 4 toistosäännön 9 mukaisesti silmukan vartijaosan predikaatti $A_i \neq x$. Valitaan variantiksi i, sillä se kuvaa etenemistä taulukossa alkio kerrallaan alkuunpäin. Näin jälkimmäinen osa muuntuu muotoon **do** $A_i \neq x \rightarrow i{:}\,[inv \wedge (A_i \neq x), inv \wedge (0 \leq i < i_0)]$ **od.**

Lopuksi johdetaan silmukan runko. Tähän sovelletaan uudelleen sijoitussääntöä [Watson and Kourie, 2012, 59], jolloin rungoksi saadaan $i = x - 1$. Jälkimmäinen osa on nyt muotoa **do** $A_i \neq x \rightarrow x = i - 1$ **od.**

Kun osat tuodaan uudelleen yhteen saadaan ohjelma

$i \coloneqq A.len - 1$
do $A_i \neq x \rightarrow$
    $x = i - 1$
od,

joka näyttäisi olevan perättäishaun toteutus. Tavoiteltavaa on, että ohjelmia johdettaessa käytettäisiin automatisoituja työkaluja eikä itse johtamista tehtäisi käsin [Smith and Hoebel, 2010]. Edellä annettu toimii kuitenkin aiheeseen perehtyvälle esimerkkinä ohjelman johtamisesta sen määrittelystä johtamissääntöjä käyttäen.

5. Yhteenveto

Correct-by-construction-menetelmät ovat formaalien ohjelmointimenetelmien joukossa mahdollisesti formaalien menetelmien käyttöön perinteisesti liitettyjen ongelmakohtien ratkaisija. Niiden, kuten formaalien menetelmien yleensäkin, käyttöönottoa ja laajempaa soveltamista ohjelmistotuotannossa rajoittaa kuitenkin formaalien menetelmien maine ja osittain ominaisuudet. [Woodcock et al., 2009]

Kyselytutkimusten [Woodcock et al., 2009] perusteella formaaleihin menetelmiin liittyvät asenteet ovat muuttumassa ja niitä työssään soveltaneet ovat ainakin jossain määrin nähneet menetelmien käytöstä saatavan edun. Correct-by-construction-menetelmät voivat ainakin osaltaan olla parantamassa formaalien menetelmien mainetta käyttökelpoisina menetelminä [Chapman, 2005].

Correct-by-construction-menetelmät ovat laitteistojen suunnittelussa ja toteuttamisessa huomattavasti ohjelmistotuotantoa käytetympiä, mutta kirjoittajan oman mielenkiinnon vuoksi menetelmien käyttö laitteistojen kontekstissa on jätetty tutkielman aihepiirin ulkopuolelle.

Viiteluettelo

[Abrial, 2006] Jean-Raymond Abrial: Formal methods in industry: achievements, problems, future. In: *Proceedings of the 28th International Conference on Software Engineering*, 761–767.

[Clarke and Wing, 1996] Edmund M. Clarke, Jeannette M. Wing et al., Formal Methods: State of the Art and Future Directions, *ACM Computing Surveys*, **28** (1996), 626–643.

[Chapman, 2006] Roderick Chapman, Correctness by Construction: A Manifesto for High Integrity Software. In: *Proceedings of the 10th Australian Workshop on Safety Critical Systems and Software*, **55** (2005), 43–46.

[Chaudhari and Om Hamani, 2015] Dipak L. Chaudhari and Om Damani, *Introducing Formal Methods via Program Derivation*. In: *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, 266–271.

[Dijkstra, 1975] Edsger W. Dijkstra, Guarded Commands, Nondeterminacy and Formal Derivation of Programs, *Communications of the ACM*, 1**8** (1975), 453–457.

[Hoare, 1969] C. A. R. Hoare, An Axiomatic Basis for Computer Programming, *Communications of the ACM*, 1**8** (1975), 576–583.

[Kourie and Watson, 2012] Derrick G. Kourie and Bruce W. Watson, *The Correctness-by-Construction Approach to Programming*. Springer Berlin Heidelberg, 2012.

[Morgan, 1998] Carroll Morgan, *Programming from Specifications.* Prentice Hall, 1998.

[Smith and Hoebel, 2010] Douglas R. Smith and Louis Hoebel, Derivational Software Engineering. In: *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, 255–358.

[Sommerville, 2007] Ian Sommerville, *Software Engineering*. Pearson Education, 2007.

[Woodcock et al., 2009] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald, Formal methods: Practice and experience. *ACM Computing Surveys*, **41** (2009), Article 19.

Junan matkustajainformaatiojärjestelmän järjestelmätestaus

Konsta Peltola

Tiivistelmä.
Tässä tutkielmassa esittelen lähijunien matkustajainformaatiojärjestelmän järjestelmätestausta. Matkustajainformaatiojärjestelmällä tarkoitetaan junassa olevista erilaisista tietojenkäsittelylaitteista koostuvaa sulautettua järjestelmää, jonka tehtävänä on välittää informaatiota junan matkustajille. Tutkielmassa perehdytään myös järjestelmätestauksen käsitteeseen ja sen rooliin osana ohjelmistokehitysprosessia. Integraatiotestauksen osalta testauksen edistymistä seurataan myös esimerkkitapausten avulla. Julkaisu- ja hyväksymistestauksesta ei ole esimerkkejä, mutta niiden suunnittelusta ja tarkoituksesta kerrotaan myös junan matkustajainformaatiojärjestelmän näkökulmasta.

Avainsanat ja -sanonnat: matkustajainformaatiojärjestelmä, sulautettu järjestelmä, järjestelmätestaus, integraatiotestaus, julkaisutestaus, hyväksymistestaus.

1. Johdanto

Junan matkustajainformaatiojärjestelmän tai lyhemmin PIS-järjestelmän (Passenger Information System) tehtävä on välittää tietoa junan matkustajille. Tämä tapahtuu muun muassa erilaisten junan sisäisten ja ulkoisten näyttöjen sekä kuulutusjärjestelmien avulla. Matkustajainformaatiojärjestelmän avulla voidaan normaalin reittiin liittyvän tiedon lisäksi viestiä myös erilaisista poikkeavista tilanteista ja ohjeistaa matkustajia. Lisäksi järjestelmään saattaa liittyä myös erilaisia matkustajien turvallisuuteen liittyviä ominaisuuksia, kuten valvontakameraja hätäpuhelinjärjestelmiä.

Järjestelmätestauksen tehtävä on varmistaa kehitettävän ohjelmiston tai tässä tapauksessa laajemman järjestelmän oikeellinen toiminta ennen sen käyttöönottoa ja toimitusta asiakkaalle. Tämä tarkoittaa, että järjestelmän tulee vastata sille asetettuja vaatimuksia ja järjestelmä on mahdollisimman virheetön. Järjestelmätestaus on koko ohjelmistokehitysprosessin ajan jatkuvan testauksen viimeinen vaihe ennen kuin järjestelmä otetaan käyttöön. [Sommerville 2007]

Matkustajainformaatiojärjestelmän järjestelmätestauksen toteuttaa erillinen testaustiimi ja suurin osa testeistä suoritetaan testiympäristössä, joka vastaa mahdollisimman monilta osin todellista tilannetta junassa. Testauksen suunnittelun ja toteutuksen apuna käytetään testauksenhallintajärjestelmää, johon on mahdollista luoda erilaisia testitapauksia ja testisarjoja, tallentaa testien tuloksia sekä tuottaa erilaisia raportteja ja koosteita testaustuloksista.

Tässä tutkielmassa esittelen lähijunien matkustajainformaatiojärjestelmän järjestelmätestausta. Tutkielmassa perehdytään myös järjestelmätestauksen käsitteeseen ja rooliin osana ohjelmistokehitystä. Integraatiotestauksen osalta testauksen edistymistä seurataan myös junan sisäisten LED-näyttöjen esimerkin avulla. Julkaisu- ja hyväksymistestauksesta ei ole esimerkkejä, mutta niihin perehdytään myös junan matkustajainformaatiojärjestelmän näkökulmasta.

Matkustajainformaatiojärjestelmän järjestelmätestausta seurataan sellaisen projektin kannalta, jossa vanhaa jo liikenteessä käytössä ollutta järjestelmää kehitettiin eteenpäin uusien ominaisuuksien avulla. Tutkielman kirjoitushetkellä olin osana kyseisen projektin testaustiimiä ja näin osallisena järjestelmän järjestelmätestauksessa. Integraatiotestauksesta esitetyt esimerkkitestit ja niiden tulokset on otettu tästä projektista.

Tämän johdannon jälkeen tutkielman toisessa luvussa perehdytään järjestelemätestauksen käsitteeseen ja osa-alueisiin sekä sen rooliin osana ohjelmistokehitystä. Tämän jälkeen kolmannessa luvussa tutustutaan testitapausten suunnittelun periaatteisiin ja erilaisiin testausmenetelmiin. Neljännessä luvussa perehdytään hieman tarkemmin raideliikenteen matkustajainformaatiojärjestelmien toimintaan ja muun muassa niitä sääteleviin standardeihin. Viidennessä luvussa esitellään matkustajainformaatiojärjestelmän integraatiotestausta lähijunan sisälle sijoitettujen LED-näyttöjen testausesimerkin avulla. Kuudennessa luvussa perehdytään julkaisu- ja hyväksymistestaukseen junan matkustajainformaatiojärjestelmän näkökulmasta. Tutkielman viimeisessä luvussa kerätään yhteen tutkielman tulokset yhteenvedon muodossa.

2. Järjestelmätestaus

Järjestelmätestauksen tehtävä on paljastaa järjestelmästä sellaisia virheitä, jotka eivät johdu järjestelmän komponenteista sellaisenaan [Beizer 1990]. Tällaiset virheet syntyvät komponenttien rajapintojen välisessä kommunikaatiossa ja komponenttien toiminnassa osana järjestelmää.

Järjestelmätestauksesta ja sen jakautumisesta eri vaiheisiin on olemassa hyvin monenlaisia tulkintoja. Glenford ja muut [2011] toteavatkin, että järjestelmätestaus on kaikkein väärinymmärretyin ja vaikein vaihe testauksessa. Suomessa oman lisänsä tulkintoihin tuovat tietysti myös erilaiset käännökset suomen kielelle, ja suomenkielisissä julkaisuissa järjestelmätestauksesta saatetaankin puhua myös suoremmin englanninkielestä käännetyllä termillä systeemitestaus.

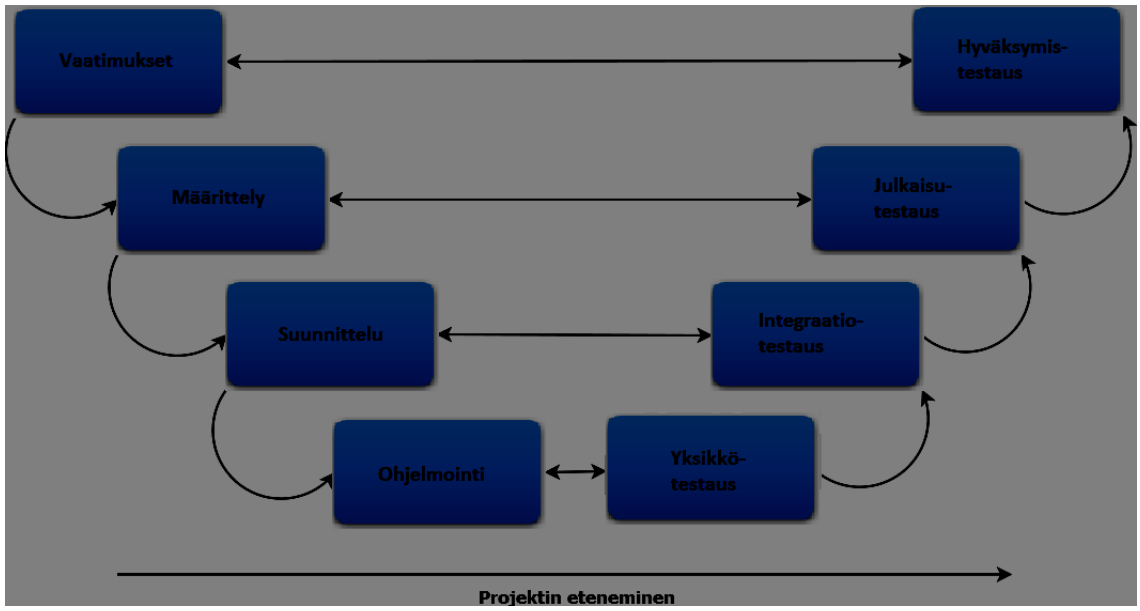
Sommervillen [2007] mukaan järjestelmätestaus jakautuu integraatiotestaukseen ja julkaisutestaukseen, joka pitää sisällään myös hyväksymistestauksen vaiheen. Monet pitävät kuitenkin integraatio-, järjestelmä- ja hyväksymistestausta täysin omina vaiheinaan ohjelmistontestauksessa [Glenford *et al.* 2011; Kasurinen 2015]. Tällöin järjestelmätestaukseksi mielletään yleensä se, mitä Sommerville kutsuu julkaisutestaukseksi, ja integraatiotestaus sekä hyväksymistestaus

erotellaan omiksi vaiheikseen. Sommervillen mukaan hyväksymistestaus kuitenkin sisältyy julkaisutestauksen vaiheeseen. Tästä johtuen suomeksi on puhuttu Sommervillen julkaisutestauksesta jopa hyväksymistestauksena. Hyväksymistestauksella tarkoitetaan kuitenkin vain julkaisutestauksen viimeistä vaihetta, jolloin myös asiakas osallistuu testaukseen. Mielestäni asiakkaan osallistuminen testaukseen on hyvä erottaa selvästi muusta julkaisutestauksesta, joten puhun tässä tutkielmassa Sommervillen termistä ”release testing” julkaisutestauksena ja pidän hyväksymistestauksen omana osanaan julkaisutestausta. Päädyin muiltakin osin käyttämään Sommervillen määritelmiä käsitteille, koska ne sopivat mielestäni hyvin junan matkustajainformaatiojärjestelmän testauksen vaiheiksi.

Erilaisista näkemyksistä käsitteiden rajauksissa on haittaa lähinnä testauksesta keskusteltaessa, koska samoista vaiheista voidaan puhua eri nimillä. Käsitteiden rajaukset onkin hyvä sopia yhdessä testaustiimin tai organisaation kesken, jotta turhilta väärinkäsityksiltä vältyttäisiin. Koko testausprosessin ymmärtämistä eri käsitteiden rajaukset eivät kuitenkaan hankaloita, koska kaikki näkemykset kuitenkin sisältävät pitkälti samat asiat, mutta vain hieman eri tavoin jaoteltuina. Käsitteiden rajauksia kannattaakin miettiä myös kehitettävän järjestelmän testaustarpeiden ja testauksen painopisteen kannalta.

2.1. Järjestelmätestauksen rooli ohjelmistokehityksessä

Testaus tulisi nähdä koko ohjelmistokehityksen ajan kestävänä prosessina. Alussa ohjelmiston vaatimuksia, rakennetta ja toimintaa suunniteltaessa tulee suunnitella myös ohjelmiston testausta, sillä mitä myöhemmin virheet järjestelmän toiminnassa huomataan, sitä kalliimpaa niiden korjaaminenkin on [Coplien and Bjornvog]. Kuvassa 1 esitetään testauksen V-malli, johon olen muokannut testauksen vaiheiden nimet niin, että ne vastaavat tässä tutkielmassa käyttämiäni rajauksia. Esittelen V-mallin, koska se sopii hyvin kuvaamaan junan matkustajainformaatiojärjestelmän kaltaisen sulautetun järjestelmän kehitystä, jossa varsinaisia julkaisuja tehdään suhteellisen harvoin.



Kuva 1. Testauksen V-malli [Kasurinen 2015].

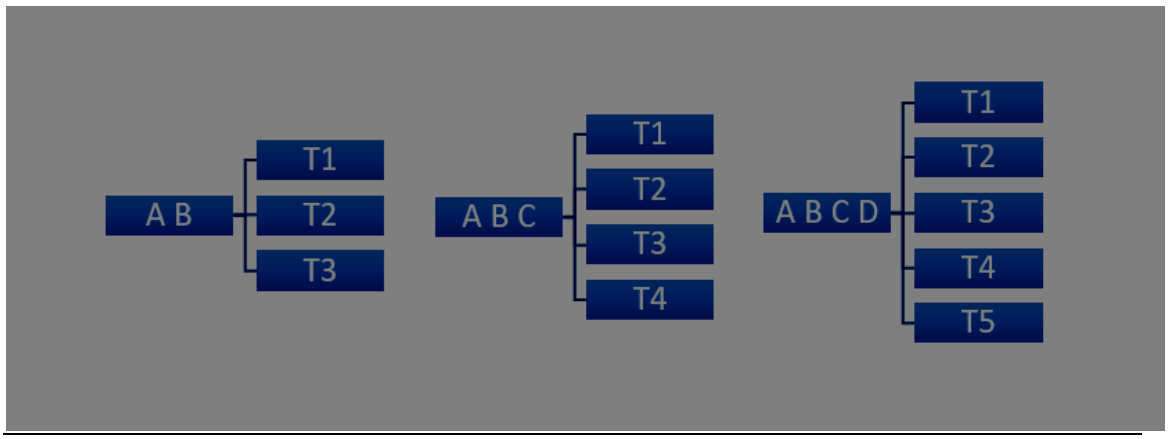
V-mallissa projekti etenee vasemmalta oikealle ja kuvan vasen puoli kuvastaa järjestelmän suunnittelua ja rakentamista ja oikea puoli järjestelmän testausta. Varsinainen testaus alkaa yksikkötesteillä, joilla testataan jotakin yksittäistä järjestelmän osaa ennen kuin se integroidaan osaksi järjestelmää. Yksikkötestauksen tekevät yleensä ohjelmoijat itse ja testattavan yksikön koko on useimmiten muutamia satoja rivejä lähdekoodia [Beizer 1990]. Tämän jälkeen vastuu testauksesta useimmiten siirtyy erilliselle testaustiimille, joka suorittaa integraatiotestauksen ja julkaisutestauksen. Hyväksymistestauksen vaiheessa testaukseen otetaan mukaan vielä asiakaskin.

V-mallia on kritisoitu paljon, koska se esittää ohjelmistokehityksen vanhan vesiputousmallin tavoin lineaarisena prosessina [Marick 1999]. V-mallia tulee kuitenkin ymmärtää niin, että jokainen vaihe sisältää iteratiivista kehitystä ja ketjussa saatetaan liikkua myös takaisinpäin.

Ohjelmoijien ja testaustiimin tulee toimia yhteistyössä ohjelmiston laadun varmistamiseksi, mutta nämä kaksi ryhmää on kuitenkin hyvä erottaa toisistaan. Erillisen testaustiimin vahvuus on se, että tuolloin testaajat eivät tunne järjestelmän tarkkaa toteutusta. Jos ohjelmoijat suorittaisivat myös järjestelmätestauksen, he saattaisivat hyödyntää tuntemustaan järjestelmän toteutuksesta ja suunnitella helposti sellaisia testitapauksia, joista järjestelmä suoriutuisi varmasti. Järjestelmätestauksen vaiheessa testaustiimi ja ohjelmoijat saattavat kuitenkin kommunikoida esimerkiksi siitä, kuinka eri ominaisuuksia voidaan testata mahdollisimman monipuolisesti. Yleensä testaustiimille pitäisi riittää tietämys integroitavan komponentin rajapinnoista. Ohjelmoijien on kuitenkin tällöinkin hyvä katselmoida testitapauksia, jotta testien kattavuus voidaan varmistaa. Testauksen edetessä, tulee testaustiimin puolestaan kommunikoida järjestelmän toiminnasta

löytämistään virheistä takaisin ohjelmoijille, jotka tekevät löydettyjen virheiden pohjalta korjauksia järjestelmään. Julkaisutestauksen vaiheessa ohjelmoijien ei kuulu enää osallistua testaukseen, vaan järjestelmää testataan loppukäyttäjän näkökulmasta. Integraatio- ja julkaisutestauksen jälkeen järjestelmä siirtyy testauksen viimeiseen vaiheeseen, hyväksymistestaukseen, jolloin asiakaskin on mukana testauksessa. Nämä testit läpäistyään järjestelmä on valmis käyttöönotettavaksi. [Kasurinen 2015].

2.2. Integraatiotestaus



Kuva 2. Integraatiotestaus [Sommerville 2007].

Integraatiotestauksen vaiheessa testataan järjestelmään lisätyn ominaisuuden toimintaa osana koko järjestelmää [Kasurinen 2015]. Kuva 2 esittää integraatiotestauksen etenemistä. Aluksi ominaisuudet A ja B sisältävälle järjestelmälle tehdään testit 1, 2 ja 3. Seuraavassa vaiheessa järjestelmään integroidaan ominaisuus C ja ajetaan vanhat testit 1,2 ja 3 sekä ominaisuutta C koskevat testit 4. Samaa kaavaa jatketaan, kunnes järjestelmä on kokonainen. Järjestelmän rakentaminen aloitetaan niin kutsutusta minimijärjestelmästä, johon ominaisuuksia lisätään yksi kerrallaan. Aina uuden ominaisuuden lisäämisen jälkeen ajetaan myös aikaisemmin lisätyille ominaisuuksille tarkoitetut testit. Näin varmistetaan, että järjestelmään aikaisemmin integroidut ominaisuudet toimivat edelleen odotetusti. Aikaisempien testien uudelleen ajamista kutsutaan regressiotestaukseksi. Järjestys, jossa ominaisuudet integroidaan järjestelmään, saattaa vaihdella rakennettavan järjestelmän mukaan. Yleisimpänä tapana pidetään kuitenkin, että ominaisuudet integroidaan niin, että kaikkein käytetyimmät ja tärkeimmät ominaisuudet integroidaan ensin. Näin tärkeimpien ominaisuuksien toimintaa voidaan testata integroinnin aikana kaikkein eniten. [Sommerville 2007]

2.3. Regressiotestaus

Regressiotestaus tarkoittaa joidenkin jo aikaisemmin suoritettujen testien uudelleen ajamista. Erityisesti tätä käytetään integraatiotestauksen vaiheessa, jolloin

uuden ominaisuuden järjestelmään lisäämisen jälkeen halutaan varmistaa, ettei lisätty ominaisuus ole rikkonut vanhaa, jo toimivaksi todettua toiminnallisuutta. Regressiotestausta käytetään myös esimerkiksi, jos järjestelmään tehdään yksittäisiä korjauksia vielä käyttöönoton jälkeen. Tässä tapauksessa regressiotestausta saatetaan kutsua myös savutestaukseksi [Kasurinen 2015]. Savutestauksessa ei kuitenkaan ajeta uudelleen kaikkia testejä, vaan juuri tätä tarkoitusta varten kehitetty, perustoiminnallisuuden testaava testisarja. Regressiotestauksessa käytetään usein apuna testiautomaatiota, jotta testien suorittamiseen kuluva aika ei kasvaisi kohtuuttomaksi [Sommerville 2007].

2.4. Julkaisutestaus

Julkaisutestauksen vaiheessa keskitytään testaamaan järjestelmän toimintaa kokonaisuutena. Varsinaiset julkaisutestit suoritetaan musta laatikko -menetelmän avulla, koska tässä vaiheessa ollaan kiinnostuneita lähinnä järjestelmän ulkoisesta toiminnasta. Musta laatikko -testauksen käsitteeseen perehdytään paremmin vielä seuraavassa luvussa. Julkaisutestauksen vaiheen alussa järjestelmää testataan yleensä vielä testiympäristössä tai simuloidusti. Todelliseen ympäristöön siirrytään vasta julkaisutestauksen lopulla, hyväksymistestauksen yhteydessä. Esimerkiksi junan matkustajainformaatiojärjestelmän tapauksessa ei järjestelmän jatkuva testaaminen junassa ole käytännössä edes mahdollista, vaan testaus on suoritettava mahdollisimman pitkään testiympäristössä. Julkaisutestauksen avulla pyritään osoittamaan, että luotu järjestelmä saavuttaa sille asetetut vaatimukset ja toimii normaalissa käytössä [Kasurinen 2015]. Julkaisutestauksen vaiheessa voidaan kuitenkin yrittää myös tahallisesti rikkoa järjestelmää antamalla sille esimerkiksi virheellisiä syötteitä.

Kun järjestelmä on kokonainen, on mahdollista testata myös sellaisia järjestelmän ominaisuuksia, jotka johtuvat järjestelmän toiminnasta kokonaisuutena, kuten sen tehokkuutta, virheistätoipumiskykyä tai nopeutta suorittaa jokin tärkeä vaatimus [Sommerville 2007]. Näitä ominaisuuksia kutsutaan myös emergenteiksi ominaisuuksiksi.

Julkaisutestaus ja siihen sisältyvä hyväksymistestaus ovat järjestelmätestauksen viimeisiä vaiheita ja jos virheitä järjestelmän toiminnassa ei huomata tässä vaiheessa, päätyvät ne valmiiseen tuotteeseen ja mahdollisesti loppukäyttäjälle asti. Vaikka näissä viimeisissä testeissä havaittavien virheiden korjaaminen saattaa olla työlästä, vaikeaa tai mahdotonta aikataulun puitteissa, on niiden löytäminen ennen käyttöönottoa kuitenkin tärkeää. Tällöin mahdollisiin ongelmiin osataan varautua tai julkaisua lykätä.

2.5. Hyväksymistestaus

Hyväksymistestaus on viimeinen osa julkaisutestausta ja tässä vaiheessa asiakaskin osallistuu järjestelmän testaukseen. Tämän testauksen vaiheen tarkoituksena on osoittaa tuotteen laatu ja kyky täyttää sille asetetut vaatimukset. Viimeistään

tässä vaiheessa järjestelmää testataan sen todellisessa käyttöympäristössä. Hyväksymistestit läpäistyään ja asiakkaalta saadun hyväksynnän jälkeen järjestelmä on valmis käyttöönotettavaksi. [Kasurinen 2015].

3. Testitapausten suunnittelu ja testaustavat

3.1. Testitapausten suunnittelu

Testitapausten suunnittelu on yksi testauksen tärkeimmistä vaiheista. Testausta voidaan nimittäin tehdä kuinka paljon tahansa ja kuinka hyvillä tuloksilla tahansa, mutta jos testien suunnittelu on tehty huonosti, saattaa suurin osa järjestelmässä olevista virheistä jäädä täysin huomaamatta. Voidaankin ajatella, että myös testit pitää testata [Beizer 1990]. Käytännössä tämä tapahtuu useimmiten projektin eri sidosryhmien suorittaman katselmoinnin avulla.

Testitapauksilla testataan järjestelmän toimintaa erilaisissa tilanteissa. Testitapauksia suunniteltaessa tavoitteena on keksiä sellaisia tapauksia, jotka ovat mahdollisimman kattavia ja vastaavat suoritettavan testauksen tavoitteita. Testauksen tavoitteena saattaa esimerkiksi olla löytää ohjelmasta virheitä, jolloin sitä voidaan testata vaikkapa virheellisillä syötteillä. Joskus tavoitteena saattaa olla vain todistaa, että ohjelma täyttää sille asetetut vaatimukset ja toimii normaalissa käytössä. Testitapauksia saatetaan tehdä kuvitteellisten käyttötapausten pohjalta ja tällöin testitapaukset pohjautuvat tilanteisiin, jollaisista järjestelmän tulee suoriutua usein. Testitapauksilla saatetaan myös hakea järjestelmän kestävyyden rajoja, kuten maksimaalista samanaikaisten käyttäjien määrää tai nopeutta suoriutua kriittisistä tehtävistä.

Testitapaukset tulisi kirjata ylös esimerkiksi testauksenhallintaohjelmaan ja kirjoittaa niin yksityiskohtaisesti, että muutkin kuin testiryhmän jäsenet ymmärtävät ne. Ohjelman avulla testejä on myös mahdollista hallinnoida helposti ja niiden tuloksia voidaan seurata. Testeistä voidaan luoda eri tilanteisiin sopivia testisarjoja ja tulokset voidaan erotella testattavan järjestelmän eri versioiden mukaan. Esimerkiksi integraatiotestauksen vaiheessa ohjelmoijat voivat testauksenhallintaohjelman kautta katselmoida testitapauksia ja ehdottaa muutoksia testien kattavuuden parantamiseksi.

Projektin alussa testitapauksia luodaan usein suoraan järjestelmän vaatimusmäärittelystä ja testit kohdistuvat järjestelmän yksittäisiin ominaisuuksiin. Testauksen edetessä voidaan suunnitella uusia testejä, jos esimerkiksi huomataan, että nykyisten testien kattavuus ei ole tarpeeksi hyvä. Vanhoja testitapauksia saatetaan myös yhdistää, jos niiden tulokset ovat olleet hyviä ja testit ovat osittain päällekkäisiä tai muuten helposti suoritettavissa samalla kerralla. Projektin alussa testitapaukset tulisi kuitenkin kirjoittaa niin, että testattavaa ominaisuutta voidaan testata mahdollisimman tarkasti, eikä esimerkiksi muiden ominaisuuksien keskeneräisyys pääse vaikuttamaan testin tuloksiin. Projektin

lopulla julkaisu- ja hyväksymistestauksen vaiheissa testitapausten tulisi perustua pääasiassa erilaisiin käyttötapauksiin. [Kasurinen 2015]

3.2. Testaustavat

Erilaisia kehityksen aikana ja ennen julkaisua käytettäviä testaustapoja on todella paljon, joten käsittelen tässä vain junan matkustajainformaatiojärjestelmän järjestelmätestauksen kannalta tärkeimpiä menetelmiä. Eri menetelmien käyttö riippuukin paljon kehitettävästä järjestelmästä ja testaamiseen suunnatuista resursseista.

Musta laatikko -testauksella tarkoitetaan testausta, jossa välitetään vain ohjelmistolle annettavista syötteistä ja siltä saatavista tulosteista. Itse ohjelmistoa pidetään ”mustana laatikkona”, jonka sisäiseen toimintaan ei perehdytä. Näin pystytään testaamaan käyttötapauksina erilaisia tilanteita, joista järjestelmän tulee suoriutua loppukäyttäjänkin käytössä. Erityisesti julkaisutestausta ja hyväksymistestausta suoritetaan musta laatikko -testauksena, koska tuolloin ollaan kiinnostuneita nimenomaan järjestelmän ulospäin näkyvästä toiminnasta. [Kasurinen 2015]

Valkoinen laatikko -testauksella puolestaan tarkoitetaan testausta, joka on suunniteltu niin, että ohjelmiston toteutus tunnetaan. Tällöin ollaan kiinnostuneita myös järjestelmän sisäisestä toiminnasta. Valkoinen laatikko -testausta saatetaan tehdä integraatio- ja julkaisutestauksenkin vaiheissa, mutta käytännössä sitä tehdään eniten yksikkötestauksen yhteydessä. Tällöin ohjelmoija tuntee järjestelmän tai sen osan toteutuksen ja pystyy luomaan testejä jotka testaavat sen koodia mahdollisimman monipuolisesti. [Kasurinen 2015] Joissain tapauksissa myös testaajien tulee tuntea järjestelmän toimintaa. Esimerkiksi kun integraatiotestauksen vaiheessa testataan järjestelmän rajapintoja, on hyvä tuntea järjestelmän toimintaa hieman syvällisemmin. Jos osalla testaajista on ymmärrystä myös järjestelmän toiminnasta, voivat he kommunikoida testeissä löytyneitä virheistäkin tarkemmin ohjelmoijille. Testaajien tulee kuitenkin aina muistaa, että testauksen tavoitteena on löytää virheitä, eikä kiertää niitä tietoisesti, vaikka ohjelmiston toteutuksen tunteminen sen joskus saattaisi mahdollistaakin.

Suorituskykytestausta voidaan tehdä koko kehitysprosessin ajan, mutta useimmiten sitä tehdään julkaisutestauksen vaiheessa, kun järjestelmä on kokonainen. Järjestelmä on enemmän kuin osiensa summa, joten tällöin on mahdollista testata koko järjestelmän emergenttejä ominaisuuksia, kuten sen tehokkuutta ja luotettavuutta. Emergentit ominaisuudet ovat yhteisiä koko järjestelmälle ja muodostuvat koko järjestelmän toiminnasta. Yksittäisten komponenttien nopeus tai tehokkuus ei nimittäin tarkoita, että niistä koottu järjestelmä olisi nopea tai tehokas. Suorituskykytestauksessakin voidaan testata järjestelmän toimintaa normaaleissa tilanteissa tai kokeilla järjestelmän kestävyyden rajoja. Järjestelmälle voidaan asettaa vaatimusmäärittelyssäkin erilaisten ominaisuuksien

lisäksi suorituskykyyn liittyviä vaatimuksia, kuten kuinka nopeasti järjestelmä on valmiina käytettäväksi uudelleen käynnistyksen jälkeen. [Sommerville 2007]

3.3. Testiautomaatio

Testiautomaatio tarkoittaa testien suorittamista erillisen ohjelmiston avulla, joka osaa verrata testien tuloksia odotettuihin tuloksiin [Sommerville 2007]. Testien automatisointi on erityisen tärkeää, kun kehitetään suuria järjestelmiä, jolloin regressiotestaus saattaa muuttua liian aikaa vieväksi ja mekaaniseksi. Automaation avulla regressiotestaukseen kuluvat resurssit saadaan minimoitua, kun testejä ei jouduta ajamaan manuaalisesti uudelleen ja uudelleen. Testiautomaatiota ei voida kuitenkaan aina soveltaa tai sen käyttö on joissain tilanteissa hyvin vaikeaa. Testitapauksia suunniteltaessa tuleekin miettiä, kuinka helposti testi olisi automatisoitavissa. Nykyisin testausautomaatiota käytetään paljon ja joidenkin järjestelmien testaus on jopa täysin automatisoitua. Aina kun järjestelmää muutetaan, lähtevät testit käyntiin automaattisesti, ja jos kaikki testit eivät mene puhtaasti läpi, ei muutos siirry myöskään ohjelmiston aktiiviseen julkaisuun. Todellisuudessa kaikkien testien automatisointi on usein mahdotonta ja testiautomaation rooli onkin useimmiten tukea manuaalista testausta ja vapauttaa testaajien aikaa mekaanisesta, erilaisten syötteiden näppäilystä luovempaan ja kokeilevampaan testaukseen.

4. Raideliikenteen matkustajainformaatiojärjestelmät

Nykyaikaiset raideliikenteen matkustajainformaatiojärjestelmät ovat sulautettuja, erilaisista tietojenkäsittelylaitteista rakentuvia järjestelmiä. Paperisia reittikarttoja ja aikatauluja on edelleen tarjolla, mutta nykyisin ne toimivat lähinnä elektronisen järjestelmän tukena. Raideliikenteen informaatiojärjestelmiin liittyykin paljon historiaa ja esimerkiksi junan näyttöjen asettamista osoittamaan reittiä kutsutaan edelleen usein junan kilvittämiseksi. Elektronisiin matkustajainformaatiojärjestelmiin kuuluvat junissa olevien järjestelmien lisäksi myös asemilla olevat näyttö- ja kuulutusjärjestelmät. Matkustajainformaatiojärjestelmien tehtävä on tehdä matkustamisesta helppoa ja miellyttävää sekä lisätä matkustamisturvallisuutta. Matkustajalle voidaan tarjota tietoa esimerkiksi aikataulujen muutoksista, matkan edistymisestä, viivästysten syistä sekä vaihtoyhteyksistä eri asemilta. Järjestelmien toimintaan halutaan panostaa, jotta asiakkaat kokisivat matkustamisen helpoksi ja käyttäisivät junaa matkustamiseen tulevaisuudessakin. Järjestelmien suunnittelussa tuleekin kiinnittää erityistä huomiota monien erilaisten käyttäjien tarpeisiin. Tietoa ei saa esimerkiksi tarjota pelkkien kuulutusten avulla, vaan visuaalisen esityksen avulla varmistetaan, että tieto saavuttaa myös huonokuuloiset matkustajat. Matkustajainformaatiojärjestelmän avulla on mahdollista esittää myös muuta materiaalia, kuten mainoksia tai uutisia, jotka saattavat toimia järjestelmän tilaajalle myös lisätulon lähteenä.

Erilaisissa junissa on hyvin erilaisia matkustajainformaatiojärjestelmiä. Esimerkiksi vanhimmissa junissa tieto reitistä tai pääteasemasta saattaa olla näkyvillä vain junan ulkopuolella. Nykyaikaiset järjestelmät ovat kuitenkin elektronisia, junan sisäiseen lähiverkkoon perustuvia kokonaisvaltaisia järjestelmiä, jotka pyrkivät pitämään matkustajan ajan tasalla koko matkan ajan. Nykyaikaiset junien matkustajainformaatiojärjestelmät voidaan jakaa karkeasti kahteen osaan, kiinteä- ja muuttuvarakenteisiin. Esimerkiksi pitkänmatkan liikenteessä käytetään yleensä junia, joiden rakenne on muutettavissa ja erilaisista vaunuista voidaan luoda loputon määrä erilaisia kokoonpanoja. Tämä luo omanlaisensa haasteen matkustajainformaatiojärjestelmälle, jonka tulee sopeutua erilaisiin kokoonpanoihin. Lähiliikenteessä puolestaan junan rakenne on yleensä kiinteä, tosin näitä kiinteitäkin yksiköitä voidaan monesti yhdistää, mutta tilanne on silti monin tavoin yksinkertaisempi.

Tässä tutkielmassa keskitytään kuitenkin vain nykyaikaiseen lähijunan matkustajainformaatiojärjestelmään. Tutkielmassa lähijunalla tarkoitetaan useista vaunuista koostuvaa, mutta rakenteeltaan kiinteää junayksikköä. Näitä junayksiköitä voidaan kuitenkin yhdistää, jolloin eri yksiköiden matkustajainformaatiojärjestelmien pitää pystyä toimimaan yhteistyössä.

4.1. Matkustajainformaatiojärjestelmä lähijunassa

Tässä tutkielmassa käsiteltävä lähijunan matkustajainformaatiojärjestelmä rakentuu erilaisista junan sisäisistä ja ulkoisista näytöistä, turvakameroista, äänentoistojärjestelmästä, hätäpuhelimista, käyttöliittymälaitteista ja keskustietokoneesta, jotka kaikki keskustelevat junan sisäisen lähiverkon välityksellä.

Junan matkustajainformaatiojärjestelmän välittämällä tiedolla on kaksi pääkohderyhmää. Ensimmäisen kohderyhmän muodostavat asemalla olevat matkustajat, joille viestitään junan ulkoisten LED-näyttöjen avulla. Asemalla olevia matkustajia kiinnostaa muun muassa tieto junan reitistä ja pääteasemasta. Asemalla oleville matkustajille tarjotaan tietoa myös asemalla olevan matkustajainformaatiojärjestelmän avulla. Toinen kohderyhmä ovat junan kyydissä olevat matkustajat. Heille välitetään tietoa muun muassa matkan edistymisestä, vaihtoyhteysmahdollisuuksista eri asemilla ja mahdollisista poikkeustilanteista. Tätä tarkoitusta varten myös junan sisällä on LED- ja TFT-näyttöjä, jotka kertovat junan reitistä. Lisäksi TFT-näytöillä voidaan näyttää tietoa saapumisajoista eri asemille, lipunmyynnistä junassa tai esimerkiksi junan nopeudesta ja ulkolämpötilasta.

Tärkeä osa järjestelmää on kuulutuslaitteisto, joka kuuluttaa automaattisesti seuraavan aseman nimen ja mahdolliset vaihtoyhteydet ennen asemalle saapumista. Kuulutuslaitteiston avulla voidaan myös antaa esimerkiksi manuaalisesti etukäteen nauhoitettuja kuulutuksia. Automaattiset ja etukäteen nauhoitetut kuulutukset voidaan näyttää kuulutuksen hetkellä myös tekstinä TFT-näytöillä.

Junan konduktööri ja kuljettaja voivat myös antaa manuaalisia kuulutuksia matkustajille puhelinjärjestelmän avulla. Puhelinjärjestelmä mahdollistaa myös junan ohjaamosta toiseen soittamisen, jolloin konduktööri ja kuljettaja voivat kommunikoida keskenään. Junan henkilökunta muodostaakin matkustajainformaatiojärjestelmän kolmannen kohderyhmän. Junan ohjaamoista löytyy käyttöliittymälaite, jonka avulla henkilökunta voi esimerkiksi asettaa junan näytöt osoittamaan reittiä tai he voivat seurata matkustamon tapahtumia turvakamerajärjestelmän avulla. Turvakamerajärjestelmään saattaa liittyä myös tallennin, joka mahdollistaa nauhoitteiden tarkastelun vielä myöhemminkin. Matkustajainformaatiojärjestelmään liittyy myös hätäpuhelinjärjestelmä, jonka avulla matkustajat saavat yhteyden junan henkilökuntaan mahdollisissa hätätilanteissa. Näin henkilökunta voi kutsua tarvittaessa lisäapua tai esimerkiksi pysäyttää junan, jos tilanne sitä vaatii. Junan matkustajainformaatiojärjestelmällä onkin tärkeä rooli myös matkustamisen turvallisuuden varmistamisessa.

4.2. Alan säännökset ja normit

Rautatieliikennettä säätelevät monet erilaiset normit ja standardit. Esimerkiksi kansainvälinen rautatieliitto UIC (International Union of Railways) vaatii jäseniltään tiettyjen standardien noudattamista. Nämä standardit käsittelevät junan teknisen toteutuksen lisäksi muun muassa tietoliikenneyhteyksien toteutusta junasta toiseen sekä junan sisäverkon kaapelointia. Standardien avulla pyritään muun muassa parantamaan turvallisuutta ja tekemään kalustosta yhteensopivaa [UIC].

Myös Euroopan unioni on asettanut omia vaatimuksiaan rautatiejärjestelmille. Vaatimuksilla pyritään siihen, että matkustajainformaatiojärjestelmät toimisivat jokseenkin samalla tavalla kaikkialla Euroopassa. Näin varmistetaan, että myös matkustajien on helppo ymmärtää eri järjestelmiä eri puolella Eurooppaa. Euroopan unionin komission tekemässä TSI-PRM-standardissa säädetään muun muassa, että junan määränpää tai reitti on esitettävä jokaisen vaunun sisäpuolella ja matkustajainformaatiojärjestelmällä on voitava antaa ilmoituksia useammalla kuin yhdellä kielellä. [EU] Matkustajainformaatiojärjestelmän kannalta myös säädökset vammaisten ja liikuntarajoitteisten esteettömän matkustamisen turvaamiseksi ovat tärkeitä. Erilaisten käyttäjien huomioiminen matkustajainformaatiojärjestelmän suunnittelussa onkin erityisen tärkeää.

4.3. Testiympäristö

Testiympäristö, jossa matkustajainformaatiojärjestelmää testataan, on rakennettu vastaaman mahdollisimman monilta osin todellista tilannetta järjestelmän käyttöympäristössä. Testiympäristössä pystytään muun muassa simuloimaan useita junayksiköitä samanaikaisesti, jolloin on mahdollista testata myös tilanteita joissa kahdella tai kolmellakin junayksiköllä ajetaan yhdessä. Testijärjestel-

mässä pystytään suorittamaan kaikki tärkeimmät testit. Näin varsinaisten junatestien, joiden järjestäminen on vaikeaa, kallista, eikä aina edes mahdollista, määrä saadaan pidettyä minimissä. Testijärjestelmällä pystytään suorittamaan testejä, jotka eivät vaadi junan liikkumista, mutta järjestelmällä on myös mahdollista suorittaa paljon erilaisia muuttuvaa dataa vaativia testejä. Tällöin kaikki järjestelmän laitteet toimivat kuten ne toimisivat oikeassa junassakin reitillä ajettaessa.

5. Matkustajainformaatiojärjestelmän integraatiotestaus

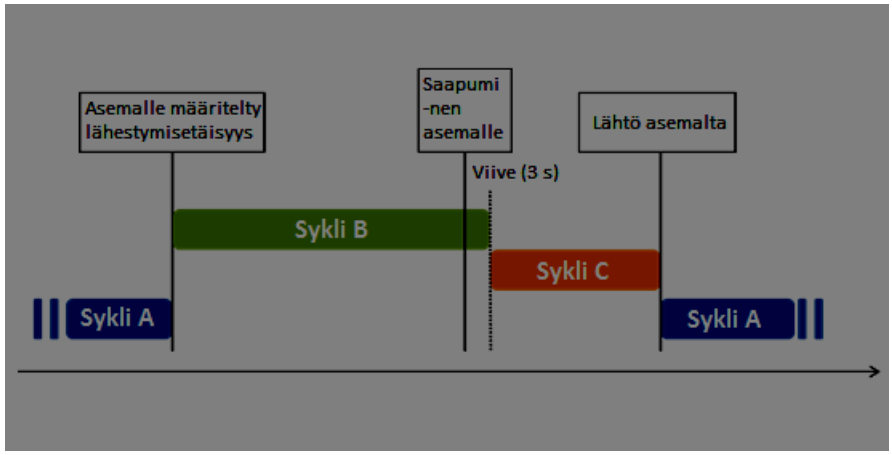
Lähijunan matkustajainformaatiojärjestelmän integraatiotestaus alkaa testitapausten määrittämisellä järjestelmän vaatimusmäärittelyn pohjalta. Tämän jälkeen kyseiset testit suoritetaan, kun testattava ominaisuus integroidaan osaksi testijärjestelmää. Käytännössä testitapausten määrittämistä ja testien suorittamista tehdään eri ominaisuuksien osalta samanaikaisesti. Jokainen järjestelmän vaatimus tulee sisällyttää testitapauksiin ja aluksi onkin hyvä luoda jokaiselle vaatimukselle oma testitapauksensa. Myöhemmin kun suurin osa testaukseen kuluvasta ajasta menee regressiotestaukseen, voidaan joitakin helposti samalla kerralla suoritettavia testejä yhdistää yhdeksi testitapaukseksi. Käytän tässä tutkielmassa esimerkkinä junan sisäisten LED-näyttöjen uuden toiminnallisuuden integraatiotestausta. Sisäisten LED-näyttöjen tarkoitus on kertoa matkustajille tietoa junan reitistä ja pääteasemasta.

5.1. Testitapausten määrittäminen

Aluksi ominaisuuksille luodaan testitapauksia järjestelmän vaatimusmäärittelyn pohjalta. Kehitystiimi implementoi järjestelmän ominaisuuksia vaatimusmäärittelyn pohjalta, joten näin saadaan helposti määriteltyä kutakin implementoitavaa ominaisuutta vastaavat testitapaukset. Projektinhallintatyökalun avulla testaustiimi pystyy seuraamaan siitä, mitkä ominaisuudet ovat seuraavaksi valmiita integroitaviksi testijärjestelmään. Tällöin testaajat osaavat luoda näille ominaisuuksille tarkemmat testitapaukset etukäteen, jotta testaus päästään aloittamaan mahdollisimman nopeasti. Testitapauksista kannattaa tehdä tarkemmat versiot vasta tässä vaiheessa, koska järjestelmän vaatimukset saattavat muuttua projektin edetessä. Näin vältytään turhalta testitapausten päivittämiseltä ja varmistetaan, että testaajilla on viimeisin tieto siitä, kuinka järjestelmän tulisi toimia ja mitkä ovat testien odotetut tulokset. Testitapaukset tulee myös muotoilla niin, että muiden järjestelmän ominaisuuksien mahdollinen keskeneräisyys tai virheellinen toiminta ei pääse vaikuttamaan testattavan ominaisuuden toimintaan tai testin tulokseen.

Kuvassa 3 esitetään junan sisäisten LED-näyttöjen suunniteltu toiminnallisuus junan reitin aikana. Usein lähiliikenteen junissa ja busseissa vastaavan näytön tehtävä on näyttää vain seuraava asema tai pysäkki, mutta tässä tapauksessa

näyttöä halutaan käyttää hieman monipuolisemmin. Tämä johtuu osittain järjestelmää koskevista säädöksistä, koska toteutettavan toiminnallisuuden avulla, junan sisäiset LED-näytöt täyttävät EU:n TSI-PRM -standardin jo yksinään. Näin TFT-näyttöjen käyttö vapautuu huomattavasti ja niillä voidaan esittää välillä esimerkiksi uutisia tai mainoksia, koska vaadittu matkustajainformaatio välittyy matkustajille jo pelkkien junan sisäisten LED-näyttöjen avulla.



Kuva 3. Junan sisäisten LED-näyttöjen toiminnallisuus.

Syklissä A, kun juna on lähtenyt asemalta, LED-näyttö näyttää vuorotellen tietoa seuraavasta asemasta ja pääteasemasta. Pääteaseman yhteydessä näytetään myös junan linjatunnus sekä jos pääteasemalle on määritelty jokin kuvake, näytetään kuvake näytön oikeassa reunassa. Tällaisia yleisesti käytettyjä kuvakkeita saattavat olla esimerkiksi laiva sataman yhteydessä tai lentokone lentokentän aseman yhteydessä. Jos seuraava asema on pääteasema, näytetään vain seuraava asema -näkymää. Syklissä B, junan lähestyessä asemaa, LED-näyttö näyttää vain seuraavan aseman nimen. Syklissä C, junan ollessa asemalla, näyttö näyttää junan pääteaseman, linjatunnuksen ja mahdollisen pääteasemalle määritellyn kuvakkeen.

Projektinhallinnassa ja vaatimusmäärittelyssä jokainen näytön toiminnallisuuden sykli on jaettu omaksi ominaisuudekseen, joten myös integraatiotestit suunnitellaan aluksi jokaista sykliä varten. Tällöin jos jonkin syklin toiminnassa ilmenee virheitä, voidaan testit kuitenkin suorittaa hyväksytysti muille sykleille. Testit myös muotoillaan niin, että niiden suorittaminen ei vaadi muiden syklien toimimista. Nämä testit voidaan kuitenkin mahdollisesti myöhemmin yhdistää yhdeksi testitapaukseksi julkaisutestauksen vaiheessa.

Testejä suunniteltaessa tulee ajatella normaalin toiminnan lisäksi kaikkia mahdollisia erikoistilanteita. Junan matkustajainformaatiojärjestelmän yhteydessä tällaisia tilanteita saattavat olla esimerkiksi näytöt ensimmäisellä tai viimeisellä asemalla, erilaiset vaatimukset eri reittilinjoilla tai talviaikaan siirtymi-

nen junien aikatauluissa. Usein testaaja saattaa löytää sellaisiakin erikoistilanteita, joissa järjestelmän toimintaa ei ole määritelty lainkaan järjestelmän vaatimusmäärittelyssä. Tällöin vastausta tulee selvittää projektin johdosta tai jopa asiakkaalta asti. Näistä erikoistilanteista tulee luoda omia testitapauksiaan alkuperäisten rinnalle, jotta kaikki mahdolliset tilanteet muistetaan testata, eikä yksittäinen erikoistilanne muuta kaikkien testien tulosta hylätyksi.

Sisäisten LED-näyttöjen varsinaisen toiminnallisuuden lisäksi vaatimuksena oli, että näyttöjen tehonkulutus ei saa missään tilanteessa ylittää tiettyä rajaarvoa. Näyttöjen kirkkautta on mahdollista muuttaa ohjelmiston avulla, joten maksimirajan ylittyminen piti estää kirkkaudensäädön avulla. Tehonkulutuksen mittausta varten luotiin myös oma testitapauksensa. Maksimaalinen tehonkulutus saadaan, kun kaikki näytön ledit palavat samaan aikaan ja kirkkaus on säädetty maksimiarvoon.

Tehonkulutuksen testaus on hyvä esimerkki testitapauksesta jossa testaajien tulee tuntea järjestelmän toimintaa. Testaajien tulee selvittää ohjelmoijien kanssa, kuinka näytön virrankulutus saadaan maksimiinsa. Testaus suoritetaan siis valkoinen laatikko -testauksena.

5.2. Testien suorittaminen

Kun uusi ominaisuus on integroitu osaksi järjestelmää, voidaan ominaisuudelle suorittaa sille suunnitellut integraatiotestit. Myös aiemmin integroitujen ominaisuuksien testit ajetaan, jotta voidaan olla varmoja, että uuden ominaisuuden integroiminen osaksi järjestelmää ei ole rikkonut vanhaa toiminnallisuutta. Testausautomaatiolla tätä regressiotestaukseen kuluvaa aikaa voidaan lyhentää.

Kun testit on suunniteltu ja sisäisten LED-näyttöjen ohjelmisto asennettu testijärjestelmään päästään testejä vihdoin suorittamaan. Kaikki kolme sisäisten LED-näyttöjen toiminnallisuuden sykliä oli jo implementoitu, joten testejä päästiin suorittamaan useampia samalla istumalla. Tässä tilanteessa näyttöjen tulisi siis toimia oikeellisesti kaikissa mahdollisissa tilanteissa. Testejä suoritettiin simulaattorin avulla testijärjestelmässä käyttäen useita erilaisia reittejä. Minkäänlaista testiautomaatiota näissä testeissä ei käytetty apuna, vaan ne suoritettiin manuaalisesti. Kuvissa 4 ja 5 näkyy sisäisten LED-näyttöjen seuraava asema- ja pääteasemanäkymät. Kuvat on otettu simuloitujen testiajojen aikana testijärjestelmässä.

Kuva 4. Seuraava asema -näkymä

Kuva 5. Pääteasemanäkymä.

Jos testit on suunniteltu ja dokumentoitu hyvin, pitäisi itse testien ajamisen olla melko suoraviivaista ja mutkatonta. Testaajan pitäisi pystyä vain lukemaan testitapauksen ohjeista mitä hänen tulee tehdä. Tämä on tavoitteena siksi, että jos samat testit suorittaakin seuraavalla kerralla joku muu niin testit suoritettaisiin kuitenkin samalla tavalla eikä testien suorittaja vaikuttaisi testien tuloksiin. Testitapauksen suorittamisen jälkeen testaaja merkkaa tapauksesta mahdolliset muistiinpanot ja tuloksen testauksenhallintajärjestelmään.

Myös LED-näyttöjen tehonkulutusta pystyttiin testaamaan testijärjestelmässä. Ohjelmoijilta saatujen ohjeiden avulla kaikki näytön ledit voitiin asettaa palamaan yhtä aikaa ja kirkkaus saatiin säädettyä maksimiin. Tämän jälkeen virrankulutus mitattiin näytön virtakaapelista ja mittausten tulokset kirjattiin testauksenhallintajärjestelmään.

5.3. Testien tulokset

Testien suorituksen jälkeen on aika arvioida niiden tuloksia. Testauksenhallintajärjestelmän avulla tuloksista voidaan generoida erilaisia raportteja eri osapuolten tarpeiden mukaan. Mahdollisesta virheistä tulee viestiä välittömästi esimerkiksi ohjelmoijille, jotta korjauksen tekeminen voidaan aloittaa mahdollisimman

pian. Testeissä saattaa ilmetä myös paljon sellaisia asioita, joita ei voida varsinaisesti luokitella virheiksi, mutta niistä tehdyt havainnot tulee välittää eteenpäin esimerkiksi projektin johdolle, jotta mahdolliset epäselvyydet järjestelmän vaatimusmäärittelyssä voidaan korjata.

Sisäisten LED-näyttöjen integraatiotestauksessa testien tulokset olivat hyviä ja näyttöjen toiminnallisuus täytti vaatimusmäärittelyssä asetetut vaatimukset. Testien yhteydessä huomattiin kuitenkin paljon sellaisia pieniä asioita, joita ei vaatimusmäärittelyssä ole mainittu lainkaan. Yhtenä esimerkkinä tällaisesta oli näyttöjen siirtyminen syklistä toiseen. Sitä, miten siirtymisen pitäisi tapahtua, ei ole määritelty missään, mutta testeissä huomattiin siirtymisen tapahtuvan hieman epäsujuvasti. Näyttö vaihtaa syklistä toiseen heti saatuaan käskyn vaihtaa sykliä. Tällöin edellinen sykli saattaa jäädä kesken. Testien ansioista vaatimusmäärittelyyn pyydetään tarkennusta ja tarvittava korjaus voidaan tehdä mahdollisimman nopeasti.

Tehonkulutukseen liittyneissä mittauksissa puolestaan huomattiin, että näytön virrankulutus saattaa näytön käynnistyksen yhteydessä olla hetkellisesti liian suuri. Testien ansiosta tähän ongelmaan saadaan korjaus. Liian suurella tehonkulutuksella saattaisi olla ikäviä seurauksia, joten on tärkeää, että kaikkien järjestelmän laitteiden virrankulutus on annettujen rajojen alapuolella.

6. Matkustajainformaatiojärjestelmän julkaisutestaus

Integraatiotestauksen jälkeen siirrytään testaamaan koko matkustajainformaatiojärjestelmän toimintaa. Kaikki julkaisuun sisällytettävät uudet ominaisuudet ja korjaukset on integroitu testijärjestelmään ja järjestelmää päästään testaamaan kokonaisvaltaisemmin. Tässä vaiheessa järjestelmän ohjelmistosta luodaan uusi julkaisu, jolle luodaan versionumero. Testauksen aikana löydettävien virheiden vakavuuden ja korjaamismahdollisuuksien mukaan päätetään, sisällytetäänkö korjaus testattavaan julkaisuun vai vasta seuraavaan. Julkaisutestaus tapahtuu musta laatikko -testauksena ja sitä varten voidaan suunnitella laajempia, kattavammin järjestelmää testaavia testitapauksia. Hyviä testitapauksia ovat esimerkiksi kuvitteelliset käyttötapaukset ja erilaiset erikoistapaukset, jotka haastavat järjestelmän toimintaa.

Matkustajainformaatiojärjestelmän tapauksessa julkaisutestauksen vaiheessa yhdistellään paljon integraatiotestauksessa käytettyjä testitapauksia ja luodaan muutamia uusia, monipuolisempia testitapauksia. Tämä tarkoittaa sitä, että eri ominaisuuksien päällekkäinen toiminta tulee testata. Junan matkustajainformaatiojärjestelmässä on paljon erilaisia ominaisuuksia jotka saattavat olla ristiriidassa keskenään, eikä toimintaa kaikissa ristiriitatilanteissa ole edes voitu määritellä järjestelmän vaatimusmäärittelyssä. Suuri osa tällaisista ristiriidoista tuleekin ratkaista kehitystiimin sisällä.

Hyvänä esimerkkinä toimii vaikkapa erityyppisten kuulutusten toiminta. Integraatiotestauksen vaiheessa on varmistuttu siitä, että kaikki erityyppiset kuulutukset toimivat itsenäisesti. Nyt julkaisutestauksen vaiheessa voidaan testata, että kuulutusten prioriteettijärjestys on oikea. Tämä tarkoittaa sitä, että esimerkiksi konduktöörin antaman manuaalisen kuulutuksen pitää kaikissa tilanteissa katkaista automaattinen kuulutus. Tätä tulee testata myös niin, että automaattinen kuulutus alkaa kesken manuaalisen kuulutuksen.

6.1. Hyväksymistestaus

Hyväksymistestauksen vaiheessa myös asiakas osallistuu järjestelmän testaukseen. Junan matkustajainformaatiojärjestelmän tapauksessa tällä tarkoitetaan varsinaisia junassa suoritettavia testejä. Käytännössä asiakas ei kuitenkaan yleensä testaa järjestelmää itse luomillaan testeillä vaan seuraa järjestelmän toimintaa, ja järjestelmän toimittaja pyrkii vakuuttamaan asiakkaalle, että järjestelmä on valmis käyttöönotettavaksi.

Ensimmäisen julkaisun hyväksymistestauksen ensimmäinen vaihe on käyttöönottotesti, jossa järjestelmän ohjelmisto asennetaan uuteen junaan ensimmäisen kerran. Nämä testit suoritetaan jo junan kasaamisen yhteydessä tehtaalla ja tarkoituksena on lähinnä varmistaa, että kaikki informaatiojärjestelmän laitteet toimivat ja kaapelointi on tehty oikein. Tehtaalla kaapelointiin on vielä mahdollista tehdä korjauksia suhteellisen helposti tai mahdollisia rikkinäisiä laitteita voidaan vaihtaa.

Seuraava tärkeä osa matkustajainformaatiojärjestelmän hyväksymistestauksessa ovat viranomaisten suorittamat testit, joilla varmistetaan, että järjestelmä täyttää normien ja standardien asettamat vaatimukset. Jos kaikkia matkustajainformaatiojärjestelmää koskevia vaatimuksia ei ole otettu toteutuksessa huomioon, selviää se melko varmasti tässä vaiheessa. Nämä viranomaisten tekemät testitkin suoritetaan yleensä vain ensimmäisen julkaisun yhteydessä.

Hyväksymistestauksen viimeinen vaihe, joka suoritetaan jokaiselle julkaisulle, ovat junalla todellisessa ympäristössä suoritettavat junatestit. Näissä testeissä on usein mukana sekä junan valmistajan että liikennöitsijän edustajia. Tässä vaiheessa kaiken tulisi toimia jo melko saumattomasti ja testien tarkoituksena on vain varmistaa, että kaikki toimii odotetusti. Tarkoituksena on myös osoittaa asiakkaille, että järjestelmä toimii oikein ja täyttää sille asetetut vaatimukset. Siirtyminen testijärjestelmästä oikeaan junaan ei kuitenkaan aina ole mutkatonta, koska joidenkin ominaisuuksien kokonaisvaltainen testaaminen ei ole mahdollista muualla kuin järjestelmän todellisessa käyttöympäristössä. Jos kaikki osapuolet ovat testien jälkeen tyytyväisiä järjestelmän toimintaan, voidaan uusi julkaisu ottaa käyttöön myös kaupallisessa liikenteessä.

7. Yhteenveto

Järjestelmätestauksella ja siitä vastaavalla testaustiimillä on tärkeä rooli järjestelmän laadunvarmistuksessa. Lisäksi matkustajainformaatiojärjestelmän tapauksessa testauksella varmistetaan, että järjestelmä täyttää sen toimintaa säätelevät normit ja standardit. Virheet on tärkeää löytää mahdollisimman aikaisin ohjelmistonkehitysprosessissa, koska tällöin niiden korjaaminen on helpompaa ja halvempaa. Hyvän testaustiimin ja hyvin suunnitellun, koko ohjelmistonkehityksen ajan kestävän testausprosessin avulla, järjestelmän laadun varmistaminen on mahdollista. Testauksen ja laadunvarmistuksen tulee kuitenkin olla osa jokaisen projektissa työskentelevän ajatusmaailmaa. Testaus vaatii paljon resursseja, mutta sen avulla saavutettu hyöty kumoaa usein siitä aiheutuneet haitat ja vaadittavia resurssejakin voidaan yrittää vähentää esimerkiksi testien automatisoinnin avulla.

Tutkielmassa esitellyssä junan sisäisten LED-näyttöjen tapauksessakin testauksella saatiin selviä tuloksia. Testien avulla järjestelmästä löydettiin virheitä sekä epäselviä kohtia, joita voidaan nyt tarkentaa ja korjata. Esimerkissä esiteltiin vain pieni osa suuren järjestelmän järjestelmätestauksesta, jonka avulla lopulta koko järjestelmän laatu voidaan varmistaa. Matkustajainformaatiojärjestelmän tapauksessa testausta vaikeuttaa kuitenkin todellisessa käyttöympäristössä suoritettavien testien vähäinen määrä ja kallis hinta.

Tässä tutkielmassa esitetty matkustajainformaatiojärjestelmän järjestelmätestausprosessi on monilta osin soveltuva muidenkin sulautettujen järjestelmien testaukseen. Prosessi esitettiin tutkielmassa niin yleisluontoisesti, että se on sovellettavissa moniin muihinkin järjestelmiin kuin vain junien matkustajainformaatiojärjestelmiin. Esimerkkien avulla kuitenkin havainnollistettiin, mitä testausprosessin eri vaiheet tarkoittavat junan matkustajainformaatiojärjestelmän tapauksessa. Nämä esimerkit ovat melko helposti muunnettavissa koskemaan muitakin sulautettuja järjestelmiä, kunhan testausprosessin eri vaiheet osataan tunnistaa.

Viiteluettelo

Boris Beizer. 1990. *Software testing techniques.* Second edition. Van Nostrad Reinhold.

James Coplien and Gertrud Bjornvig. 2010. *Lean Architecture for Agile Software Development.* Wiley.

EU. 2008. Komission päätös L 62/74. *Euroopan unionin virallinen lehti, 7.3.2008.*

Jussi Pekka Kasurinen. 2015. *Ohjelmistotestauksen käsikirja.* Docendo.

Brian Marick. 1999. New Models for Test Development. http://www.exampler.com/testing-com/writings/new-models.pdf. Checked 11.12.2015.

Glenford Myers, Tom Badgett and Corey Sandler. 2011. *The Art of Software Testing.* Wiley.

Ian Sommerville. 2007. *Software Engineering.* Pearson Education.
UIC. 2015. International Union of Railways. www.uic.org. Checked 10.12.2015.

Antipatternit ja antipatternien havainnointi

Atte Pietikäinen

Tiivistelmä.
Antipatternit ovat huonoja toimintatapoja, joilla on dokumentoitu parempi ratkaisu. Vaikka antipatternit ovat lähes yhtä tärkeitä ohjelmistokehityksen kannalta kuin suunnittelumallit, ei antipatterneja ole tutkittu läheskään yhtä laajalti kuin suunnittelumalleja. Suurin osa tutkimuksesta on keskittynyt antipatternien havainnointiin, johon on määritelty useita eri menetelmiä.

Avainsanat ja -sanonnat: Antipatternit, antipatternien havainnointi

1. Johdanto

Suunnittelumallit ovat eräitä tärkeimpiä tapoja lähestyä ongelmia ohjelmistotuotannossa. Kaikki toimintatavat eivät ole yhtä hyödyllisiä kuin suunnittelumallit. Toimintatavat, joiden vaikutus ei ole hyödyllinen vaan ohjelmistolle haitallinen, kutsutaan antipatterneiksi. Brownin ja muiden [1998] mukaan yleisesti käytettyä toimintatapaa voidaan kutsua antipatterniksi, jos sillä on havaittava haitallinen vaikutus, ja jos sille on dokumentoitu parempi vaihtoehto.

Toisin kuin suunnittelumalleja, antipatterneja ei ole tutkittu laajalti. Suuri osa tutkimuksesta keskittyy antipatternien havainnointiin sekä antipatternien vaikutuksiin kooditasolla. Vuoteen 2009 mennessä ei antipatterneja ollut analysoitu lainkaan suunnittelutasolla. [Llano and Pooley 2009]

Antipatternit voidaan jakaa kolmeen kategoriaan: kehityksellisiin, rakenteellisiin ja johtamisantipatterneihin. Kehitykselliset antipatternit ovat ratkaisuja ohjelmoijien kohtaamiin ongelmiin, rakenteelliset antipatternit järjestelmän rakenteellisiin ongelmiin ja johtamisantipatternit ohjelmistokehityksen ja organisoitumisen ongelmiin. [Brown *et al.* 1998]

Tässä tutkielmassa keskityn kehityksellisiin antipatterneihin ja niiden havainnointiin. Luvussa 2 esittelen Brownin ja muiden määrittelemät seitsemän normaalia antipatternia ja luvussa 3 seitsemän mini-antipatternia. Luvussa 4 käsittelen erilaisia antipatternien havainnointimenetelmiä.

2. Normaalit antipatternit

Tässä luvussa käsittelen Brownin ja muiden [1998] määrittelemät ohjelmoinnissa kohdattavat normaalit antipatternit. Esittelen millaisia antipatternit ovat, mitä vaikutuksia niillä on ja kuinka antipatternit voidaan korjata eli refaktoroida. Brown ja muut [1998] määrittelevät neljätoista eri kehityksellistä antipatter-

nia, jotka voidaan jakaa normaaleihin antipatterneihin ja mini-antipatterneihin. Normaaleilla antipatterneilla on usein laajempi vaikutus kuin mini-antipatterneilla. Normaaleja kehityksellisiä antipatterneja ovat jumalaluokka, laavavirta, funktionaalinen hajottaminen, poltergeist, kultainen vasara, spagettikoodi ja leikkaa–liimaa-ohjelmointi.

2.1. Jumalaluokka

Jumalaluokka on olio-ohjelmoinnissa esiintyvä antipatterni, jossa yksi luokka tekee suurimman osan prosessoinnista ja muut luokat sisältävät vain dataa. Jumalaluokka-antipatternissa data ja prosessit tai funktiot eritellään toisistaan. Jumalaluokan luokitellaan proseduraalisiin malleihin, vaikka se määritellään olioiden avulla ja oliopohjaisilla ohjelmointikielillä. [Brown *et al.* 1998]

Jumalaluokka on usein seurausta ohjelmiston rakenteellisista ongelmista. Ohjelmistosta voi esimerkiksi puuttua oliopohjainen rakenne tai rakenne voi puuttua kokonaisuudessaan. Jumalaluokka voi myös esiintyä tilanteissa, joissa suunniteltua rakennetta ei ole noudatettu. [Brown *et al.* 1998]

Iteratiivisissa ohjelmistoprojekteissa ohjelmoijat voivat usein lisätä toimiviin luokkiin lisää toiminnallisuuksia eivätkä luo uusia luokkia näille toiminnallisuuksille. Tämä myös voi johtaa jumalaluokan luontiin. Ohjelmiston vaatimukset voivat myös johtaa jumalaluokan esiintymiseen koodissa. [Brown *et al.* 1998]

Jumalaluokka tekee koodista vaikealukuisemman erityisesti silloin, kun jumalaluokka esiintyy yhdessä spagettikoodin kanssa. [Abbes *et al.* 2011] Abbes ja muut eivät havaitse tilastollisesti merkittäviä muutoksia pelkän jumalaluokan sisältävän koodin ymmärtämisessä, mutta tutkimus osoittaa koodin ymmärtämisessä olevan muutoksia.

Jumalaluokka on usein liian monimutkainen käytettäväksi uudelleen tai testattavaksi. Se myös vie paljon resursseja koneelta, koska pienimmänkin prosessin suorittaminen vaatii koko luokan lataamista koneen muistiin. [Brown *et al.* 1998]

Jumalaluokalle parempi ratkaisu on hyvinkin yksinkertainen. Jumalaluokka tulee hajottaa pienempiin alaluokkiin, jotka jakavat jumalaluokan toiminnallisuuden. Myös dataa sisältävät luokat voivat ottaa osan jumalaluokan toiminnallisuuksista, jotka muokkaavat tai käyttävät kyseisen luokan dataa. [Brown *et al.* 1998]

Eräs poikkeustapaus, jossa jumalaluokan käyttö on hyväksyttävää, on uuden rajapinnan luonti vanhentuneelle legacy-ohjelmistolle. [Brown *et al.* 1998]

2.2. Laavavirta

Laavavirta on kokoelma vanhaa dokumentoimatonta koodia, jota on hyvin vaikea ymmärtää. Laavavirran pystyy tunnistamaan koodin kommenttien puutteesta, tärkeältä näyttävistä funktioista, joiden tarkoitus ei ole selkeä, ja käyttä-

mättömästä koodista, joka on jätetty ohjelmistoon ilman syytä. [Brown *et al.* 1998]

Laavavirran yleinen aiheuttaja on prototyyppien koodin päätyminen tuotantoon. Koska prototyyppeihin kirjoitettu koodi on usein dokumentoimatonta, hätäisesti kirjoitettua ja kommentoimatonta, sen asettaminen suoraan tuotantoon ilman muutoksia aiheuttaa ongelmia. Prototyyppien koodi voi usein päätyä tuotantoon, koska kehittäjien on pidettävä kiinni demonstraatioiden aikarajoista tavoitteiden muuttuessa. [Brown *et al.* 1998]

Yksittäinen ohjelmoija voi myös saada aikaan laavavirran. [Brown *et al.* 1998] Yksin työskentelevä ohjelmoija voi jättää koodin kommentoimatta, koska hän olettaa, että kukaan muu ei tule käyttämään tai muokkaamaan kyseistä koodia.

Laavavirran aiheuttajille on usein yhteistä juuri koodin dokumentaation ja kommentoinnin puuttuminen. Koodin ymmärtäminen on vaikeampaa ilman dokumentaatiota ja kommentteja. Tästä johtuen laavavirran koodia ei usein pystytä muokkaamaan ja laavavirta pystyy kasvamaan uusien kehittäjien työstäessä laavavirrallista ohjelmistoa. [Brown *et al.* 1998]

Laavavirtaa voi olla vaikea purkaa, jos se on jo päässyt koodiin. Järjestelmä tulee tutkia ja laavavirrasta määrittää, mitkä osat ja funktiot ovat käytössä sekä mitä voidaan poistaa. Kuolleen koodin poiston jälkeen järjestelmään voi ilmestyä virheitä, joita ei tule korjata heti vaan järjestelmän rakenne tulee ensin korjata. Laavavirran purkaminen vie paljon aikaa ja usein vain kokeneimmat ohjelmoijat pystyvät sen tekemään. [Brown *et al.* 1998]

Paras ratkaisu laavavirralle on ennaltaehkäisy. Ennen kuin järjestelmän koodia aletaan kirjoittaa järjestelmän rakenne tulee olla määritelty. Rakenteen määritelmää tulee myös noudattaa ja valvoa. [Brown *et al.* 1998]

2.3. Funktionaalinen hajottaminen

Funktionaalinen hajottaminen on oliopohjaisissa ohjelmointikielissä esiintyvä antipatterni, jossa ohjelmisto suunnitellaan ja kirjoitetaan sivuuttamaan luokkahierarkia ja täten oliopohjaisuus. [Brown *et al.* 1998]

Ohjelmiston, jossa on funktionaalinen hajottaminen -antipatternit, osia on hyvin vaikea käyttää uudelleen. Lisäksi tällaisen ohjelmiston selkeä dokumentointi ja ohjelmiston toiminnan selittäminen on vaikeaa, koska luokkakaaviot eivät ole loogisia. [Brown *et al.* 1998]

Funktionaalisella hajottamisella on kaksi perussyytä: kokemattomuus ja rakenteelliset ongelmat. Ohjelmoija, jolla ei ole kokemusta oliopohjaisesta ohjelmoinnista, voi käyttää funktionaalista hajottamista kirjoittamaan ohjelmiston oliopohjaisella kielellä muistuttamaan strukturoidulla kielellä kirjoitettua ohjelmistoa. Ohjelmiston rakenne voi olla määritelty käyttämään funktionaalista hajottamista, koska rakenteen määrittely on tehty ennen ohjelmiston vaatimus-

ten analyysiä. Funktionaalinen hajottaminen voi johtua myös ohjelmiston rakenteen valvonnan puutteesta. [Brown *et al.* 1998]

Funktionaalisen hajottamisen refaktorointi alkaa ohjelmiston koodiston analysoinnista. Analyysin perusteella luodaan malli, joka sisältää kaikki välttämättömät ohjelmiston osat. Mallin ulkopuolelle jäävät pienet luokat sulautetaan välttämättömiin tai niistä yhdistetään malliin sopiva suurempi luokka. Mallin ulkopuoliset luokat, jotka eivät sisällä ohjelmiston tilaan liittyvää tietoa, voidaan yrittää kirjoittaa uudelleen pelkiksi funktioiksi.

2.4. Poltergeist

Poltergeist on luokka, jolla on vähäinen vaikutus ohjelmistoon ja lyhyt elinikä. Koska poltergeistit ovat monimutkaisia ja vaikeita ymmärtää, niiden ylläpitäminen on vaikeaa. Poltergeistit ovat huonoja kolmesta pääsyystä [Brown *et al.* 1998]:

1.Poltergeistit ovat turhia ja täten tuhlaavat resursseja.

2.Poltergeistit eivät ole tehokkaita, koska ne käyttävät turhia reittejä

3.Poltergeistit vaikeuttavat kunnollisen oliopohjaisen mallin toteuttamista.

Poltergeist saattaa ilmaantua rakenteellisten ongelmien takia. Ohjelmoijat ja suunnittelijat eivät ehkä ymmärrä oliopohjaista rakennetta tai rakenne on päätetty ennen vaatimuksien määrittelyä. [Brown *et al.* 1998]

Väärä ohjelmointikielen valinta voi myös aiheuttaa poltergeisteja. Oliopohjainen ohjelmointikieli ei välttämättä ole oikea valinta jokaiseen ohjelmistoon. [Brown *et al.* 1998]

Poltergeistin refaktorointi on yksinkertaista: poltergeist poistetaan järjestelmästä ja sen toiminnallisuus korvataan joko lisäämällä sen funktiot olemassa oleviin luokkiin tai jättämällä funktiot luokkien ulkopuolelle. [Brown *et al.* 1998]

2.5. Kultainen vasara

Kultainen vasara on ohjelmoijan työkalu, kuten ohjelmointiympäristö, kirjasto tai ohjelmointikieli, jota hän pyrkii käyttämään jokaiseen uuteen projektiin vaikka se ei välttämättä olisikaan oikea vaihtoehto. Kultaisen vasaran käyttäjät eivät usein halua opetella uusia lähestymistapoja, vaikka toiset lähestymistavat käyttäisivätkin vähemmän resursseja. [Brown *et al.* 1998]

Kultainen vasara antipatternina vaikuttaa enemmän itse ohjelmoijiin kuin ohjelmistoihin. Olemassa olevat ohjelmistot ja kirjastot määrittelevät uusien ohjelmistojen ja kirjastojen rakenteen, mikäli kultainen vasara -antipatterni on läsnä, koska ohjelmoijat eivät halua lopettaa kultaisen vasaran käyttöä. [Brown *et al.* 1998]

Ohjelmoijan työkalusta voi muodostua kultainen vasara, jos ohjelmoija pystyy käyttämään kyseistä työkalua erityisen hyvin. Yhtiön aiemmat investoinnit

tiettyihin työkaluihin voivat tehdä kyseisistä työkaluista kultaisia vasaroita, koska yhtiö haluaa käyttää näitä työkaluja ja saada mahdollisimman paljon irti investoinnistaan. [Brown *et al.* 1998]

Koska kultainen vasara on enemmän ohjelmiston rakentajien psykologinen kuin ohjelmiston rakenteellinen ongelma, on sitä ratkaistaessa keskityttävä näihin psykologisiin ongelmiin. Ohjelmoijien ja hallinnon on oltava valmiita hakemaan uusia ratkaisuja eikä vain käyttää vanhoja toimivia keinoja. Myös uuden henkilöstön palkkaaminen voi auttaa kultaisen vasaran refaktoroinnissa, koska uudella henkilöstöllä ei välttämättä ole samoja mieltymyksiä kultaiseen vasaraan. [Brown *et al.* 1998]

2.6. Spagettikoodi

Spagettikoodi on ohjelmisto tai sen osa, jonka rakenne on epäselvä jopa sen kirjoittajalle. Koodi on siis nimensä mukaisesti kuin spagettia. Spagettikoodi ei usein ole uudelleenkäytettävää, koska ohjelmiston osia ei ole selkeästi eroteltu toisistaan ja ne ovat usein täysin riippuvaisia toisistaan. [Brown *et al.* 1998]

Spagettikoodin ylläpitäminen vaatii paljon resursseja ja usein spagettikoodi päätyy nopeasti pisteeseen, jossa ei ole järkevää ylläpitää koodia, vaan kirjoittaa se täysin uudelleen. [Brown *et al.* 1998]

Romano ja muut [2012] analysoivat spagettikoodin sekä yhdentoista muun antipatternin vaikutusta lähdekoodimuutoksiin 16 eri Java-pohjaisessa järjestelmässä. Spagettikoodin lisäksi ComplexClass, joka on vähintään yhden monimutkaisen ja laajan funktion sisältävä luokka, ja linkkuveitsi, joka on monia erilaisia toisiinsa liittymättömiä toimintoja sisältävä luokka, vaikuttavat siihen, kuinka paljon lähdekoodimuutoksia tehdään enemmän kuin muut antipatternit. Tämän lisäksi Romano ja muut havaitsevat että muutokset ohjelmiston toiminnallisuuteen, rajapintoihin, tietokantalauseisiin ja ehtolauseisiin tapahtuvat yleisemmin luokissa, joissa spagettikoodi esiintyy.

Abbes ja muut [2011] eivät havainneet tilastollisesti merkittäviä muutoksia koodin ymmärrettävyydessä. Jumalaluokat, joissa esiintyy myös spagettikoodia, ovat huomattavasti vaikeammin ymmärrettäviä. Tosin eräät spagettikoodiluokat vaikuttivat olevan helpommin ymmärrettäviä kuin luokat, joihin spagettikoodi ei vaikuttanut. Abbesin ja muiden mukaan tämä tapaus vaatii lähempää tarkastelua.

Paras tapa refaktoroida spagettikoodi on ennaltaehkäisy. Koodin rakenteen määrittely ennen sen kirjoittamista vähentää spagettikoodin esiintymisen mahdollisuutta huomattavasti. [Brown *et al.* 1998]

Jos spagettikoodi on jo koodistossa, Brown ja muut [1998] määrittelevät viisi vaihetta koodin siivoamiseen:

1. Getterien eli funktioiden, jotka hakevat luokan muuttujan, ja setterien eli funktioiden, jotka asettavat arvon luokan muuttujalle, luonti luokkien muuttujille ja koodin uudelleenkirjoittaminen näitä käyttäen
2. Koodisegmenttien muokkaaminen uudelleenkäytettäviksi funktioiksi
3. Funktioiden parametrien uudelleenjärjestäminen yhdenmukaisesti funktioiden välillä
4. Käyttämättömän koodin poisto
5. Luokkien, funktioiden ja muuttujien uudelleennimeäminen vastaamaan standardia.

2.7. Leikkaa–liimaa-ohjelmointi

Leikkaa–liimaa-ohjelmointi on koodin uudelleenkäyttämistä kirjoittamalla tai kopioimalla sama koodinpätkä useaan kertaan ohjelmistoon. Koodinpätkissä voi olla pieniä muutoksia muuttujiin, mutta toiminta on lähes identtistä. [Brown *et al.* 1998]

Leikkaa–liimaa-ohjelmointi -antipatternin vaikutta ohjelmiston ylläpidon kustannuksiin kasvattamalla niitä. Yhden virheen korjaaminen uudelleenkäytetyssä koodissa vaatii koko ohjelmiston läpikäymistä, koska virhe ei ole vain yhdessä paikassa. Lisäksi antipatterni liioittelee ohjelmiston rivien määrää, jolloin koodin ymmärtäminen vaikeutuu. [Brown *et al.* 1998]

Yleisin syy leikkaa–liimaa-ohjelmoinnille on ohjelmoijan kokemattomuus. Uudet ohjelmoijat usein kopioivat esimerkkikoodin ja muokkaavat sitä. Vaikka uudelleenkäytettävän yleisen komponentin kirjoittaminen olisi resurssien kannalta järkevämpää, eivät uudet ohjelmoijat välttämättä edes tiedä, kuinka tällaisia komponentteja käytetään. [Brown *et al.* 1998]

Leikkaa–liimaa-ohjelmointi -antipatternin refaktorointi on kolmevaiheinen prosessi. Refaktorointi aloitetaan tunnistamalla kaikki toiminnon kopioidut koodinpätkät. Tunnistamisen jälkeen toiminnosta luodaan yleinen versio ja tunnistetut ilmentymät korvataan käyttämään tätä uutta versiota. Lopuksi organisaatiossa otettaan käyttöön leikkaa–liimaa-ohjelmoinnin kieltävät käytännöt, joiden noudattamista tulee valvoa. [Brown *et al.* 1998]

3. Mini-antipatternit

Mini-antipatternit ovat antipatterneja, joiden vaikutus on pienempi kuin normaalien antipatternien. Brown ja muut [1998] määrittelevät seitsemän kehityksellistä mini-antipatternia: jatkuva vanhentuminen, epäselvä näkökulma, ankkuri, umpikuja, input kludge, miinakenttä ja sienihallinnointi. Tässä luvussa esittelen nämä seitsemän mini-antipatternia.

3.1. Jatkuva vanhentuminen

Brown ja muut [1998] määrittelevät jatkuvan vanhentumisen mini-antipatternina, jossa ohjelmistoista julkaistaan jatkuvasti uusia versioita. Jatkuvan vanhentumisen takia kehittäjien on vaikea pitää kaikki ohjelmistot ajan tasalla ja yhteensopivien ohjelmistojen löytäminen on vaikeaa.

Jatkuvan vanhentumisen estämiseen tarvitaan avoimet standardit, joita noudatetaan yleisesti. Ohjelmistot muuttuvat vähemmän todennäköisesti, jos ne ovat standardien mukaisia, ja tällöin uudet versiot ovat todennäköisemmin yhteensopivia. [Brown *et al.* 1998]

3.2. Epäselvä näkökulma

Epäselvä näkökulma -antipatternissa oliopohjaisen mallin näkökulmaa ei ole määritelty. Oliopohjaisten mallien perusnäkökulma voi olla vähiten hyödyllinen näkökulma. Sekoitetut näkökulmat eivät salli rajapintojen erottamista niiden toteutuksesta. [Brown *et al.* 1998]

Epäselvän näkökulman refaktorointi vaatii näkökulman määrittelyn. Oliopohjaisessa mallissa on kolme ensisijaista näkökulmaa [Brown *et al.* 1998]:

1. Bisnesnäkökulma, joka keskittyy käyttäjän malliin informaatiosta ja prosesseista
2. Määritelmänäkökulma, joka keskittyy rajapintoihin
3. Toteutusnäkökulma, joka keskittyy olioiden sisäiseen toteutukseen.

3.3. Ankkuri

Ankkuri on nykyiselle projektille hyödytön ohjelmisto tai laitteisto, jonka hankkiminen on usein kuluttanut paljon resursseja. Ankkurin hankkiminen on tuntunut hankinnan aikana järkevältä, mutta jonkin ajan ja resurssien kulutuksen jälkeen se osoittautuu hyödyttömäksi. [Brown *et al.* 1998]

Kaikille ohjelmistoille ja laitteille tulisi olla varalla jokin toinen ratkaisu, joka vaatii minimaalisen määrän ohjelmiston uudelleen työstämistä. Tämä menettelytapa vähentää mahdollisten ankkurien vaikutusta. [Brown *et al.* 1998]

3.4. Umpikuja

Umpikujassa uudelleenkäytettävään komponenttiin on tehty muutoksia ja komponentin tehnyt taho ei enää tue tai ylläpidä muunneltua komponenttia. Muutoksien jälkeen komponentin ylläpito- ja tukivastuut siirtyvät muutokset tehneelle taholle. Tällaiset muutokset voivat lyhyellä tähtäimellä nopeuttaa kehitysprosessia, mutta viimeistään, kun komponentin uusi versio julkaistaan, prosessi tulee hidastumaan. [Brown *et al.* 1998]

Umpikuja voidaan välttää kokonaan välttämällä suorien muutoksien tekeminen uudelleenkäytettäviin komponentteihin. Kun tarvittavat muutokset pystytään pitämään komponentin ulkopuolella, niitä ei tarvitse tehdä uudestaan

komponentin uuteen versioon ja komponentin tukeminen ja ylläpito säilyvät sen tekijöiden vastuulla. [Brown *et al.* 1998]

3.5. Input kludge

Input kludge on ohjelmisto, jossa käyttäjän syötettä ei käsitellä oikein. Käyttäjän syöte voi kaataa tai muuten jumittaa ohjelmiston tai syötteellä voi olla eitoivottu seuraus. Input kludge usein esiintyy, jos käyttäjän syöte on vapaata tekstiä ja ohjelmistoa ei ole testattu tarpeeksi tarkasti. [Brown *et al.* 1998]

Input kludgen välttämiseksi ohjelmisto tulee testata automatisoitua testausalgoritmia käyttäen. Testausalgoritmien tulee testata ohjelmisto odotettujen syötteiden lisäksi odottamattomilla syötteillä, koska juuri odottamattomilla syötteillä on odottamattomia tuloksia. [Brown *et al.* 1998]

3.6. Miinakenttä

Lähes kaikkien ohjelmistojen koodissa esiintyy virheitä. Näitä virheitä esiintyy moderneissa järjestelmissä lähes yhtä paljon kuin miinoja miinakentässä. Usein näillä virheillä ei välttämättä ole paljoakaan vaikutusta, mutta tärkeissä järjestelmissä pienelläkin virheellä voi olla katastrofaaliset vaikutukset kuten miinaan astumisella. Tätä runsasta virheiden määrää ja potentiaalisesti katastrofaalisia vaikutuksia kutsutaan miinakenttä-antipatterniksi. [Brown *et al.* 1998]

Miinakentän purkuun tarvitaan paljon investointia ohjelmistotestaukseen. Virheettömän ohjelmiston julkaiseminen vaatii ohjelmiston runsasta testausta. Vaaditun testausohjelmiston tekeminen voi vaatia enemmän aikaa ja resursseja kuin itse julkaistavan ohjelmiston tekeminen, mutta tällainen ohjelmisto vaaditaan tuotettavan ohjelmiston virheettömyyden takaamiseksi. [Brown *et al.* 1998]

3.7. Sienihallinnointi

Sienihallinnointi-antipatternilla kuvataan tilannetta, jossa ohjelmiston kehittäjät eristetään ohjelmiston käyttäjistä. Kehittäjille annetaan ohjelmiston vaatimukset välikäsien kautta ja näiden vaatimusten oletetaan olevan muuttumattomia. Käyttäjät eivät useinkaan pysty määrittelemään ohjelmiston vaatimuksia tarpeeksi tarkasti ilman kehittäjien apua, minkä vuoksi kehittäjät eivät pysty näiden vaatimusten perusteella tekemään ohjelmistoa, joka täyttäisi käyttäjien toiveet. [Brown *et al.* 1998]

Parempi ratkaisu sienihallinnoinnille on iteratiivinen kehitystapa, jossa käyttäjät testaavat ohjelmiston uudet iteraatiot ja antavat niistä palautetta. Tämän palautteen pohjalta ohjelmistoa muokataan seuraavaa iteraatiota varten ja tämä sykli jatkuu, kunnes ohjelmisto täyttää käyttäjien vaatimukset. Tämä toimintatapa vähentää ohjelmiston hylkäämisen riskiä. [Brown *et al.* 1998]

4. Antipatternien havainnointi

Antipatternien havainnointiin on useita eri keinoja. Vaikka manuaalinen koodin tutkiminen on yksi tapa havainnoida antipatterneja, ei se ole nykyisien massiivisien ohjelmistojen kohdalla järkevää. Koska nykyiset ohjelmistot voivat olla tuhansia tai miljoonia rivejä pitkiä, tarvitaan automatisoituja menetelmiä antipatternien havainnointiin. Suuri osa antipatterneihin liittyvästä tutkimuksesta on tehty erilaisista havainnointimenetelmistä.

Tässä luvussa käsittelen neljä menetelmää antipatternien havainnointiin järjestelmien koodista. Nämä neljä ovat Stoianovin ja Soran [2010] logiikkaan perustuvat Prolog-säännöt, Maigan ja muiden [2012] tukivektorikoneisiin perustuva SMURF, palvelukeskeisten rakenteellisten antipatternien havainnointiin keskittynyt SOMAD [Nayrolles *et al.* 2013] sekä Marinescun [2004] metriikkaan perustuvat havainnointistrategiat.

4.1. Prolog-säännöt

Prolog on eräs ensimmäisistä logiikkapohjaisista ohjelmointikielistä. Stoianov ja Sora [2010] ehdottavat Prolog-pohjaisen lähestymistavan sopivan antipatternien ja suunnittelumallien havaitsemiseen.

Suunnittelumalleille Prolog-sääntöjen luominen on helpompaa kuin antipatterneille, koska suunnittelumalleille on olemassa selkeät UML-pohjaiset kuvaukset, kun taas antipatternien kuvaukset ovat usein vain tekstiä. [Stoianov and Sora 2010]

Stoianov ja Sora [2010] määrittelevät Prolog-säännöt seitsemälle eri antipatternille, joihin kuuluvat luvussa 2 esitellyt antipatternit jumalaluokka ja poltergeist. Näiden lisäksi antipatterneille dataluokka, call super, vakio rajapinta, torjuttu rajapinta ja jojo-ongelma on Prolog-säännöt.

Stoianov ja Sora [2010] suorittivat kokeilun Prolog-sääntöjensä testaamiseksi. Tässä kokeilussa ei esiintynyt lainkaan virheellisiä esiintymiä, mutta kokeilu suoritettiin vain kuudelle eri järjestelmälle, eikä kokeilussa otettu huomioon, kuinka montaa antipatternien esiintymää säännöt eivät havainneet.

4.2. SMURF

SMURF on tukivektorikoneisiin pohjautuva antipatternien havainnointimenetelmä. Tukivektorikoneen toiminta perustuu sille ennalta annettujen näytejoukkoihin, joiden perusteella kone luokittelee sille annettavat uudet näytteet. SMURF on nelivaiheinen menetelmä, jonka vaiheet ovat seuraavat [Maiga *et al.* 2012]:

1. Harjoitusjoukkojen määrittely etsittävillä antipatterneilla
2. Tukivektorikoneen opettaminen harjoitusjoukkojen avulla
3. Datajoukon luonti järjestelmästä, jota halutaan tutkia, samoja metriikkoja käyttäen kuin harjoitusjoukkoja määriteltäessä

4. Tulosten tarkastus, uuden datan lisääminen harjoitusjoukkoihin. Vaiheet 2–4 voidaan toistaa.

Maiga ja muut [2012] kokeilevat SMURF:n tarkkuutta ja vertasivat sitä DETEX- [Moha *et al.* 2009] ja BDTEX-havainnointimenetelmiin [Khomh *et al.* 2011]. DETEX on koodihajuihin perustuva antipatternien havainnointimenetelmä, kun taas BDTEX on GQM-pohjainen (Goal, Question, Metric) havainnointimenetelmä.

Maiga ja muut [2012] vertaavat DETEX- ja SMURF-menetelmiä suoraan, mutta BDTEX-menetelmän kanssa he käyttävät Khomhin ja muiden [2011] dataa vertailussa. Koska BDTEX-menetelmä on todennäköisyyksiin perustuva kun taas SMURF tunnistaa antipatterneja suoraan, ei näitä kahta voida verrata suoraan.

Maiga ja muut [2012] vertasivat menetelmien tehokkuutta kolmessa järjestelmässä ja neljän antipatternin tunnistuksessa. Nämä kolme järjestelmää ovat avoimet Java-pohjaiset järjestelmät ArgoUML v0.19.8, Azureus v2.3.0.6 ja Xerces v2.7.0. Tunnistettavat antipatternit ovat jumalaluokka, funktionaalinen hajottaminen, spagettikoodi ja linkkuveitsi. [Maiga *et al.* 2012]

Tutkimuksen mukaan SMURF on huomattavasti parempi havaitsemaan jumalaluokan esiintymisiä järjestelmän osissa kuin DETEX, ja lisäksi SMURF havaitsee enemmän jumalaluokan esiintymisiä koko järjestelmässä kuin DETEX. Verrattaessa BDTEX:iin SMURF osoittautui tarkemmaksi ja vakaammaksi. [Maiga *et al.* 2012]

Maiga ja muut [2012] eivät esitä DETEX:n kohdalla tuloksia muista kuin jumalaluokka-antipatternista ja BDTEX:n kohdalla vain jumalaluokka- ja spagettikoodi-antipatterneista. Nämä muut tulokset voivat vaikuttaa siihen, millaisia johtopäätöksiä tästä tutkimuksesta voidaan tehdä.

Koska SMURF pohjautuu tukivektorikoneisiin, se tarvitsee jaoteltua dataa tukivektorikoneen kouluttamiseen. Maiga ja muut [2012] väittävät, että on helpompaa jaotella antipatternien esiintymiä kuin määritellä sääntöjä antipatternien havainnointiin. [Maiga *et al.* 2012]

4.3. SOMAD (Service Oriented Mining for Antipattern Detection)

SOMAD on palvelukeskeisten järjestelmien antipatternien havainnointiin keskittynyt havainnointimenetelmä. SOMAD perustuu koneoppimiseen tietolouhinnan avulla. Tiedonlouhinnalla SOMAD tutkii järjestelmässä esiintyvien palvelujen assosiaatioita ja näiden sekä annettujen metriikkojen avulla havaitsee järjestelmässä esiintyviä antipatterneja. SOMAD:n edeltäjä SODA (Service Oriented Detection for Antipatterns) vaatii kaksinkertaisen analyysin eikä pysty analysoimaan järjestelmää ilman pääsyä järjestelmän palvelujen rajapintoihin. [Nayrolles *et al.* 2013]

SOMAD:ssa on viisi erillistä vaihetta [Nayrolles *et al.* 2013]:

1. Palvelukeskeisten antipatternien assosiaatiosäännöt kuvaavien metriikkojen asettaminen järjestelmään
2. Antipatternien määrittely vaiheen 1. metriikkojen avulla
3. Havainnointiin käytettävien algoritmien luonti antipatternien määrittelyn pohjalta
4. Järjestelmän palvelujen assosiaatioiden louhinta
5. Antipatternien havainnointi järjestelmän palveluista.

Nayrolles ja muut [2013] kokeilevat SOMAD:n toimivuutta kahdessa palvelukeskeisessä järjestelmässä ja vertaavat tuloksia SODA:n vastaaviin tuloksiin. Testauksessa etsittiin kuutta eri palvelukeskeisten järjestelmien antipatternia. SOMAD osoittautui tarkemmaksi ja nopeammaksi kuin SODA. SOMAD voidaan myös mahdollisesti käyttää oliopohjaisissa järjestelmissä. [Nayrolles *et al.* 2013]

4.4. Havainnointistrategiat

Havainnointistrategia on metriikkoihin perustuva havainnointimenetelmä, jolla pystytään havaitsemaan antipatternien lisäksi muita ohjelmointivikoja ja myös suunnittelumalleja. Havainnointistrategia on ohjelmointivian tai suunnittelumallin piirteiden metriikkoihin perustuva kokoelma sääntöjä, joita voidaan käyttää havaitsemaan kyseiset viat tai mallit ohjelmiston osista. Jokaiselle vialle ja mallille täytyy luoda oma havainnointistrategia. [Marinescu 2004]

Havainnointistrategian luominen neljävaiheinen prosessi [Marinescu 2004]:

1. Vian tai mallin piirteiden, jotka voidaan havaita yhtä metriikkaa käyttämällä, arkikielisten sääntöjen hajottaminen osiin
2. Metriikoiden valinta vaiheessa 1. määritellyille sääntöryhmille
3. Suodattamismekaniikan valinta valituille metriikoille
4. Saatujen sääntöjen yhdistäminen loogisilla operaattoreilla.

Havainnointistrategioita voidaan käyttää manuaalisesti tai automatisoidulla järjestelmällä. Marinescu [2004] testaa havainnointistrategioiden toimivuutta kymmenellä eri ohjelmistovialla, joista eräs on jumalaluokka-antipatterni. Havainnointistrategioiden käyttö oli automatisoitu ja tulokset tarkistettiin manuaalisesti. Havainnointistrategiat havaitsivat vikoja 50-81% tarkkuudella ja useissa tapauksissa tämä tarkkuus paranee, jos otetaan huomioon myös strategian havaitsemat muut kuin kyseisellä strategialla etsittävät viat. [Marinescu 2004]

5. Yhteenveto

Antipatternit ovat toimintatapoja, joilla on negatiivinen vaikutus ja joille on selkeästi dokumentoitu parempi vaihtoehto. Antipatterneja ei ole tutkittu lä-

heskään yhtä paljon kuin suunnittelumalleja, vaikka ne ovat lähes yhtä tärkeitä. Tässä tutkielmassa esitellyt antipatternit eivät ole läheskään kaikki kirjoittamisen hetkellä määritellyt antipatternit, vaan mukaan on otettu Brownin ja muiden [1998] esittelemät ohjelmistokehityksessä kohdattavat antipatternit.

Antipatternien tunteminen on tärkeää ohjelmoijille, koska yleinen syy antipatternien esiintymiselle on ohjelmoijan tietämättömyys. Myös laiskuus on eräs mahdollisista syistä. [Brown *et al.* 1998]

Antipatterneilla on useita negatiivisia vaikutuksia. Näihin vaikutuksiin kuuluvat liiallinen resurssien käyttö, koodin ymmärtämisen vaikeus, ohjelmiston ylläpitämisen vaikeutuminen ja muita. Näiden takia antipatternien tunnistaminen ja refaktorointi on tärkeää. [Brown *et al.* 1998; Abbes *et al.* 2011; Romano *et al. 2012*]

Ennaltaehkäisy ja ohjelmiston rakenteen määrittely ovat parhaimpia keinoja antipatternien refaktorointiin, mutta koodin muokkaaminen on myös mahdollinen tapa refaktoroida. Usein ennaltaehkäisy ei ole mahdollista, koska nykyiset projektit odottavat tuloksia nopeasti. [Brown *et al.* 1998]

Antipatternien havainnointiin on monia eri menetelmiä. Jokaisella menetelmällä on omat vahvuutensa ja heikkoutensa, ja ne ovat usein säädetty havainnoimaan tiettyjä antipatterneja. Koska antipatternit ovat usein kuvailtu tekstinä, niiden kuvauksia ei voida käyttää suoraan havainnointiin. Useat menetelmät ovat tapoja muuntaa antipatternien kuvaukset loogisiksi ehdoiksi.

Kirjallisuuskatsaukseni aikana huomasin neljä selkeää antipatterneihin liittyvää tutkimuskysymystä:

1. Onko X antipatterni?
2. Mitä vaikutuksia antipatternilla X on?
3. Mikä on antipatterni X:n refaktoroitu ratkaisu?
4. Kuinka havaita antipatterni X?

Koska antipatternit on määritelty olevan huonoja toimintatapoja, joilla on määritelty parempi ratkaisu, antipatternien tutkimus ei vaikuta olevan laajaa. Tästä samasta syystä väitän antipatternien olevan osa ohjelmoinnin historiaa ja niiden opettaminen uusille ohjelmoijille on erittäin tärkeää, koska kuten George Santayanan sanonta kuuluu: ”Ne, jotka eivät muista historiaa, ovat tuomittuja toistamaan sitä.”

Viiteluettelo

Marwen Abbes, Foutse Khomh, Yann-Gaël Guéhéneuc and Giuliano Antoniol. 2011. An empirical study of the impact of two antipatterns, blob and spaghetti code, on program comprehension. In: *Proc. of the 15th European Conference on Software Maintenance and Reengineering (CSMR)*, 181-190.

William J. Brown, Raphael C. Malveau, Hays W. McCormick and Thomas J. Mowbray. 1998. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis.* Wiley.

Foutse Khomh, Stephane Vaucher, Yann-Gaël Guéhéneuc and Houari Sahraoui. 2011. BDTEX: A gqm-based bayesian approach for the detection of antipatterns. *Journal of Systems and Software* 84, 4, 559-572.

Maria Teresa Llano and Rob Pooley. 2009. UML specification and correction of object-oriented anti-patterns. In: *Proc. of the Fourth International Coference on Software Engineering Advances (ICSEA)*, 39-44.

Abdou Maiga, Nasir Ali, Neelesh Bhattacharya, Aminate Sabané, Yann-Gaël Guéhéneuc and Esma Aimeur. 2012. SMURF: a SVM-based incremental anti-pattern detection approach. In: *Proc. of the 19th Working Conference on Reverse Engineering (WCRE),* 466-475.

Radu Marinescu. 2004. Detection strategies: metrics-based rules for detecting design flaws. In: *Proc. of the 20th IEEE International Conference on Software Maintenance*, 350-359.

Naouel Moha, Yann-Gaël Guéhéneuc, Laurence Duchien and Anne-Françoise Le Meur. 2009. DECOR: A method for the specification and detection of code and design smells. *TSE* 36, 1, 20-36.

Mathieu Nayrolles, Naouel Moha and Petko Valtchev. 2013 Improving SOA antipatterns detection in service based systems by mining execution traces. In: *Proc. of the 20th Working Conference on Reverse Engineering (WCRE)*, 321-330.

Daniele Romano, Paulius Raila, Martin Pinzger and Foutse Khomh. 2012. Analyzing the impact of antipatterns on change-proneness using fine-grained source code changes. In: *Proc. of the 19th Working Conference on Reverse Engineering (WCRE),* 437-446.

Alecsander Stoianov and Iona Sora. 2010. Detecting patterns and antipatterns in software using prolog rules. In: *Proc. of International Join Conference on Computational Cybernetics and Technical Informatics (ICCC-CONTI),* 253-258.

Terveyssovellukset ja niiden käyttäjäkokemus

Emma Salminen

Tiivistelmä.
Tämän tutkielma on kirjallisuuskatsaus terveyssovelluksiin ja niiden käyttäjäkokemukseen. Teknologisten innovaatioiden myötä mobiililaitteiden käyttömahdollisuudet laajenevat, minkä vuoksi niistä on tullut etenkin terveyssovellusten otollisin kohdealusta. Käytettävyydellä ja käyttäjäkokemuksella on tässä konseptissa tärkeä rooli, sillä terveyssovelluksia käyttävät myös henkilöt, joilla ei ole kokemusta mobiililaitteista tai joilla on hyvin vähän kokemusta teknologiasta ylipäätään. Aihe on ajankohtainen, sillä sosiaali- ja terveydenhuollon palvelurakenne on uudistumassa, välimatkat palveluntarjoajien luokse pitenevät ja uusille etähoitomahdollisuuksille on kysyntää. Kaava tehokkaisiin terveysinterventioihin tulisi löytää mahdollisimman nopeasti, sillä muun muassa ylipainoon liittyvät ongelmat uhkaavat jo koko maailman terveyttä.

Avainsanat ja -sanonnat: Käyttäjäkokemus, käytettävyys, mobiilisovellus, terveyssovellus.

1. Johdanto

Älypuhelinten ja muiden mobiililaitteiden käytön yleistyminen on muuttanut viestintätapojamme ja mobiilisovellukset tukevatkin jo monien ihmisten jokapäiväisiä askareita [Angulo and Ferre 2014]. The Statistic Portal -sivuston [2015] mukaan Google Play -kauppa tarjoaa käyttäjilleen jo yli 1,6 miljoonaa mobiilisovellusta. Applen App Store on toiseksi suurin sovelluskauppa noin 1,5 miljoonalla sovelluksella [The Statistic Portal 2015].

Vuonna 2014 älypuhelimia käytti noin 1,44 miljardia ihmistä. Luvun ennustetaan lähes kaksinkertaistuvan seuraavien kolmen vuoden aikana, jolloin vuonna 2018 älypuhelimia käyttäisi jo noin 2,56 miljardia ihmistä. [The Statistic Portal 2015.]

Samaan aikaan elämäntavoista aiheutuvat terveysriskit, esimerkiksi ylipaino, fyysinen passiivisuus ja stressi, vaikuttavat yhä useampaan ihmiseen. Terveydenhuollon resurssit eivät kuitenkaan riitä riskeihin puuttumiseen yksilöllisellä tasolla, jolloin motivoinnin tärkeys muilla tavoin korostuu. [Ahtinen *et al.* 2009.] Siksi ei olekaan ihme, että viime vuosina älypuhelimia ja tabletteja on alettu käyttää myös terveysinterventioiden alustana mobiilisovellusten muodossa [Klasnja and Pratt 2012].

Useimmille ihmisille elämäntapojen muuttaminen on kuitenkin pitkä ja haastava prosessi, joka vaatii runsaasti henkilökohtaista motivointia. Mobiilisovellukset saattavat olla voimakas keino puuttua terveysriskeihin, sillä niiden välityksellä käyttäjälle voidaan antaa ajoittain esimerkiksi erilaisia kehotuksia tai opastusta. Jotta käyttäjä saisi sovelluksesta parhaan mahdollisen avun, tuen ja motivaation, olisi tärkeää, että sovelluksen suunnittelussa otettaisiin huomioon monet hyödyllisyys- ja käytettävyystekijät. Hyvä käyttäjäkokemus koostuu monesta eri osa-alueesta, esimerkiksi hyvästä käyttöliittymästä ja käyttäjän tarpeisiin vastaamisesta. Lisäksi hyvä käyttäjäkokemus tarjoaa miellyttäviä kokemuksia yleisesti. [Ahtinen *et al.* 2009.]

Sovellusmarkkinoiden kasvaessa kilpailu lojaaleista käyttäjistä ja heidän mielenkiintonsa herättämisestä on kiivasta. Tällöin käyttäjäkokemuksen merkitys korostuu entisestään ja juuri se tekeekin eron menestyvän ja menestymättömän sovelluksen välille. [Angulo and Ferre 2014.]

Tämän tutkielman tarkoitus on kartoittaa terveyssovellusten käyttäjäkokemuksen kehittämistä ja arviointia.. Tutkielman luvussa 2 tarkastellaan käyttäjäkokemusta ensin yleisesti, tarkentaen mobiilisovellusten käyttäjäkokemukseen ja sen kehittämiseen. Luvussa 3 kerrotaan terveyssovelluksista, niiden tulevaisuudesta ja niihin liittyvästä kritiikistä. Luvussa 4 selvitetään terveyssovellusten käyttäjäkokemuksen erikoispiirteitä ja luvussa 5 paneudutaan sovellusten käyttäjäkokemuksen arviointimenetelmiin. Luku 6 on tapaustutkimus MeeDoc-mobiilisovelluksen käyttäjäkokemuksesta ja luku 7 on tutkielman yhteenveto.

2. Käyttäjäkokemus

2.1. Määritelmä

Käyttäjäkokemus on laaja käsite, jolle on annettu vuosien varrella monia erilaisia määritelmiä [Mirnig *et al.* 2015]. Kansainvälisen standardoimisjärjestön (ISO-standardin) mukaan käyttäjäkokemuksella tarkoitetaan henkilön käyttämän tuotteen, järjestelmän tai palvelun käytöstä aiheutuvia käsityksiä ja reaktioita. Standardissa huomautetaan, että käyttäjäkokemus sisältää kaikki käyttäjän tunteet, uskomukset, mieltymykset, käsitykset, fyysiset ja psyykkiset reaktiot, käyttäytymisen ja saavutukset, jotka ilmaantuvat käyttöä ennen, käytön jälkeen tai käytön aikana. Lisäksi standardissa määritellään käyttäjäkokemuksen olevan muun muassa seurausta brändiin ja brändäykseen liittyvistä asioista, kuten esimerkiksi brändin esittelystä. [ISO DIS 9241-210.]

Joskus käyttäjäkokemus ja käytettävyys sekoitetaan toisiinsa. Käytettävyys on kuitenkin käsitteenä paljon suppeampi kuin käyttäjäkokemus [Robinson *et al.*

2014]. Käytettävyyden määritelmä on siten myös vakiintuneempi kuin käyttäjäkokemuksen määritelmä. Yksi käytettävyyden tunnetuimmista määritelmistä onkin Nielsenin [1993] määritelmä, jonka mukaan käytettävyys koostuu viidestä eri osa-alueesta; opittavuudesta, tehokkuudesta, muistettavuudesta, virheiden siedosta ja tyytyväisyydestä.

On siis selvää, että käyttäjäkokemus sisältää paljon enemmän kuin pelkkä käytettävyys. Aikaisemmin emme kantaneet tietokoneita mukanamme päivittäin, jolloin pelkkä käytettävyys riitti mittaamaan tuotteen tasoa. Nykyään kuitenkin kannamme mukanamme tai pidämme yllämme erilaisia laitteita jatkuvasti. Tällöin pelkän käytettävyyden arviointi ei enää riitä, vaan tarvitsemme uusia tehokkaita tapoja ohjeistaa ja arvioida tehokasta suunnittelua. [Robinson *et al.* 2014.]

Hyvään käyttäjäkokemukseen pyrittäessä tulisi suunnittelijoiden siis ottaa huomioon myös käyttäjien tunnereaktiot ja kehittää piirteitä, joilla on todellista merkitystä tai arvoa käyttäjälle tämän jokapäiväisessä elämässä [Robinson *et al.* 2014].

2.2. Mobiilisovellusten käyttäjäkokemus

Mobiililaitteiden käyttö ilmentää hyvin ihmisen ja koneen välistä vuorovaikutusta, sillä sitä tapahtuu kaikkialla koko ajan. Mobiililaitteiden vuorovaikutuksen tutkimus onkin erilaisten näkökulmien ja teorioiden törmäyskenttä, jossa käyttäjäkokemusta suunniteltaessa tulee ottaa huomioon myös laitteen käytöstä aiheutuvat käytännön haasteet. [Nakhimovsky *et al.* 2009.]

Mobiilisovellukset muodostavatkin siis aivan oman alueensa käyttäjäkokemuksen suunnittelussa. Vuorovaikutus tulisi suunnitella siten, että käyttötoimintojen suoritusaika olisi lyhempi mobiilisovelluksessa kuin samojen käyttötoimintojen suoritusaika tietokonetta käytettäessä. Mobiilisovelluksen toimintojen pitää siis olla mahdollisimman yksinkertaisia, mutta silti hyvin tarkkoja. Toimintojen tulisi olla helppoja ja vaatia mahdollisimman vähän painalluksia tai näppäilyä. Samanaikaisesti kuitenkin laitteet itsessään asettavat alustakohtaisia rajoituksia ja vaatimuksia käyttäjäkokemuksen suunnittelulle. [Kuusinen and Mikkonen 2014].

Erilaisilla käyttömahdollisuuksilla ja näytön koolla on vaikutusta sovellussuunnitteluun; myös käytetyllä alustalla on omat suosituksensa. Mobiilisovelluksen käyttäjäkokemusta suunniteltaessa täytyy siis huomioida sekä käyttäjät, että käytetyn alustan rajoitukset ja mahdollisuudet. Käytetyn käyttöjärjestelmän tyyliopasta seuraamalla saadaan aikaan sujuva vuorovaikutus. Se ei kuitenkaan yksin riitä tuottamaan hyvää käyttäjäkokemusta, jos sovellus ei vastaa käyttäjän tarpeisiin. [Kuusinen and Mikkonen 2014.]

2.3. Mobiilisovellusten käyttäjäkokemuksen kehittäminen

Nykyään markkinoilla on monia erilaisia mobiililaitteita ja -alustoja. Sen vuoksi sovelluskehityksen haasteena onkin monialustaisuus. Sovellussuunnittelijoiden täytyisi siis pystyä kehittämään sovelluksia, jotka toimivat saumattomasti erilaisilla laitteilla ja alustoilla. Jotta kaikki potentiaaliset markkinat saataisiin katettua, jokaiselle kohdealustalle on kehitettävä oma sovellusversionsa. [Angulo and Ferre 2014.]

Kuusinen ja Mikkonen [2014] korostavat, että mobiilisovellusten kehittäminen eroaa yleisestä järjestelmäkehityksestä ainakin seuraavilta osin:

1) Mobiilisovellusten toimintojen tulee olla keskitettyjä, kaikki ylimääräinen tulee jättää pois. Tästä johtuen johdonmukaisen käyttäjäkokemuksen luominen jo projektin alussa on tärkeää.

2) Mitä suurempi projekti on, sitä enemmän käyttäjäkokemussuunnittelun tulee olla varsinaista sovelluskehitystä edellä.

3) Mobiilialustoilla on omat yksityiskohtaiset tyylioppaansa käyttöliittymien ja vuorovaikutuksen suunnittelua varten, mistä johtuen käyttäjäkokemussuunnittelijan rooli saattaa olla kevyempi mobiilisovellusprojekteissa. Jos suunnittelija kuitenkin on perehtynyt alustan tyylioppaaseen, saattaa se parantaa sovelluksen käyttäjäkokemusta.

4) Mobiilisovellusprojektit ovat tyypillisesti pieniä, jolloin sovellus halutaan saada markkinoille nopeasti. Sovellukset pyritään kehittämään usein monille alustoille samanaikaisesti. Projektin osa-alueiden ulkoistaminen tai jakaminen voi kuitenkin aiheuttaa merkittäviä kuluja ja kasvattaa kehitykseen tarvittavaa aikaa. Siksi mobiilisovelluksia kehitettäessä saattaa järkevin vaihtoehto olla sovelluksen suunnittelu ensin vain yhdelle alustalle.

Hyvän käyttäjäkokemustason voi kuitenkin säilyttää myös monialustaisessa kehityksessä, jos kehitys tehdään mukautetusti jokaisen alustan ehtoja soveltaen. Tällöin projektissa on oltava mukana käyttäjäkokemuksen asiantuntija. Paremman käyttäjäkokemuksen saa kuitenkin todennäköisemmin, jos monien tyylioppaiden sijaan kehityksessä keskitytään vain natiivikoodin vuorovaikutuskysymyksiin. [Angulo and Ferre 2014.] Natiivikoodilla tarkoitetaan alustan omaa ja alkuperäistä ohjelmointikieltä.

Lisätutkimuksia kuitenkin tarvitaan mobiilisovellusten käyttäjäkokemuksen suunnitteluun liittyvistä kysymyksistä teoriakehyksen luomiseksi. Tutkimuksen tekeminen on tärkeää, sillä huono käyttäjäkokemus merkitsee koko tuotteen laadun alenemista. [Angulo and Ferre 2014.]

3. Terveyssovellukset

3.1. Määritelmä

Terveyssovelluksilla tarkoitetaan sovelluksia, jotka tarjoavat terveyteen liittyviä palveluja älypuhelimille tai muille mobiililaitteille [TechTarget 2011]. Mobiililaitteiden käytön lisääntyessä myös terveyssovellusten käyttö on lisääntynyt, ja suurimmat sovelluskaupat (Google Play ja Apple App Store) tarjoavatkin jo kymmeniä tuhansia terveyteen liittyviä sovelluksia [Zapata *et al.* 2015].

Terveyssovellukset voidaan karkeasti jakaa kahteen ryhmään. Ensimmäisen ryhmän muodostavat vapaasti sovelluskaupoista ladattavissa olevat kaupalliset sovellukset, joiden avulla käyttäjät voivat esimerkiksi tehdä terveellisempiä valintoja jokapäiväisessä elämässään [TechTarget 2011]. Erilaiset päivittäistä aktiivisuutta mittaavat sovellukset ovat suosittuja tässä kategoriassa.

Toisen ryhmän taas muodostavat terveydenhuollon apuvälineeksi suunnitellut sovellukset, jotka Yhdysvaltain elintarvike- ja lääkevirasto (FDA) [2015] määrittelee sovelluksiksi, jotka ovat joko terveydenhuollon laitteen tai tarvikkeen lisävaruste tai muuttavat mobiilialustan sellaiseksi. Mobiilialusta siis muuttuu lääkärin ja potilaan väliseksi kommunikointivälineeksi. Tällaisia laitteita voidaan käyttää esimerkiksi tehostamaan diabetespotilaan hoitoa [Ajana El Khaddar *et al.* 2012]. Tämän ryhmän sovelluksia saatetaan kutsua yleisesti myös käsitteellä ”mHealth”, joka tulee englanninkielen sanoista mobile health [European Commission 2015]. Euroopan komission [2015] mukaan käsite mHealth kattaa myös mobiililaitteen käytön terveyden ja hyvinvoinnin palveluissa sekä näihin liittyvässä tiedotuksessa. Käsitettä käytettäessä ei siis välttämättä ole kyse ainoastaan mobiilisovelluksista.

3.2. Langattoman yhteyden mahdollisuudet

Terveyssovelluksia on siis monenlaisia; toiset on suunnattu suoraan kuluttajille ja toiset ammatilliseen käyttöön [TechTarget 2011]. Ilman langattomien yhteyksien mahdollisuuksia terveyssovellukset eivät kuitenkaan toisi arkeemme juurikaan lisäarvoa.

Kaikenlaisilla terveyssovelluksilla on yhteinen tavoite: ne yrittävät jollain tapaa edistää käyttäjänsä ja sitä kautta koko yhteiskunnan terveyttä. Langattomien teknologioiden avulla voidaankin tehokkaasti tuottaa erilaisia terveyspalveluita käytettäväksi missä tahansa [Kumar *et al.* 2010]. Jo käytössä olevia mobiiliteknologioita ovat 2G-, 3G- ja 4G-verkot, jotka tarjoavat terveydenhuollolle monia etuja. Teknologioiden avulla lääkäri voi esimerkiksi tarkastella potilaan labo-

ratoriotuloksia tai hoitohistoriaa missä ja milloin tahansa, jolloin terveydenhuollon resurssien käyttö tehostuu ja hoidon laatu paranee. [Ajana El Khaddar *et al.* 2012.]

Sekä terveydenhuollon palveluntarjoajien että näitä palveluja tarvitsevien tahojen toiminnassa paikalla ja ajalla on iso merkitys. Avuntarvitsijat ovat usein fyysisesti kaukana palveluntarjoajista, jolloin tiedon pitää liikkua nopeasti. [Ajana El Khaddar *et al.* 2012.] Tämä on ajankohtainen aihe Suomessa tällä hetkellä, sillä sosiaali- ja terveydenhuoltoalan palvelurakennetta uudistetaan. Soteuudistuksessa on muun muassa suunniteltu, että päivystävien sairaaloiden määrä vähenisi nykyisestä yhdeksästätoista kahteentoista. Tällöin joillakin sotealueilla matka palveluntarjoajan luokse pitenisi merkittävästi. [Yle uutiset 2015.] Nämä rajoitteet voidaan kuitenkin jossain määrin rikkoa hyvien tietoliikenneteknologioiden avulla. Nykyään onkin mahdollista siirtää ääntä, videota ja muuta dataa reaaliajassa missä ja milloin tahansa. [Ajana El Khaddar *et al.* 2012.]

3.3. Tulevaisuus

Täsmällisen informaation saatavuus on terveydenhuollossa erittäin tärkeää. Terveydenhuollon henkilökunnan tulee päästä käsiksi ajantasaiseen lääketieteelliseen tietoon ajasta ja paikasta riippumatta. Siksi terveydenhuoltoala onkin sopiva ala sulautetun tekniikan sovelluksille. [Patrick *et al.* 2008.] Sulautettu tekniikka tarkoittaa huomaamattomasti toimivaa, ympäristöönsä täysin sulautuvaa kaikkialla vaikuttavaa tietotekniikkaa [Wikipedia 2013].

Sulautetun terveydenhuollon tarkoituksena on tekniikan avulla luoda ympäristö, jossa terveydenhuollon palvelut olisivat kaikkialla kaikkien saatavilla. Tämä vaatii tekniikkaa, joka sulautuu virheettömästi päivittäiseen elämäämme. [Patrick *et al.* 2008.]

Jo olemassa olevilla tekniikoilla on omat rajoituksensa terveyssovelluksiin liittyen [Ajana El Khaddar *et al.* 2012], mutta onneksi mobiilisovellusten pohjana käytettävä tekniikka kuitenkin kehittyy koko ajan tehden sovelluksista entistä parempia, nopeampia ja edullisempia [Patrick *et al.* 2008]. Tulevaisuudessa tullaankin ottamaan käyttöön niin sanottuja seuraavan sukupolven tekniikoita (next generation technologies) [Kumar *et al.* 2010], joiden avulla langaton tekniikka ulottuu kaikille markkina-alueille mahdollistaen yhteydet missä ja milloin tahansa. Tulevaisuuden teknologia integroi jo olemassa olevat tekniikat yhdeksi saumattomaksi alustaksi, joka tulee takaamaan kuvan, äänen ja tiedonsiirron laadun. Tällöin myös sovellusten siirto-, yhteensopivuus- ja liikkuvuusongelmat tulevat vähenemään. [Ajana El Khaddar *et al.* 2012.]

Terveydenhuollossa langattomat tekniikat voidaan yhdistää langattoman anturiverkon (Wireless Sensor Networks) mahdollisuuksiin, jolloin pienten (puettavien) anturien avulla on mahdollista saada terveystietoa esimerkiksi verenpaineesta ja sykkeestä. Joidenkin laitteiden avulla on myös mahdollista saada tietoa potilaan olinpaikasta. Nämä teknologiat yhdistettynä kevyisiin laitteisiin, kuten puhelimiin ja tabletteihin, mahdollistavat terveydenhuollon henkilökunnan tarkastella ja päivittää potilaiden tietoja saumattomasti missä ja milloin tahansa. Tällöin myös hoidon laatu ja potilaiden tyytyväisyys paranevat esimerkiksi palvelun nopeutumisen ansiosta. [Ajana El Khaddar *et al.* 2012.]

Mobiililaite voi tulevaisuudessa toimia myös kehon alueen verkoston (body area network, BAN) keskipisteenä. Puettavien laitteiden tai antureiden avulla voidaan mitata kehosta terveyteen liittyviä arvoja, kuten esimerkiksi glukoosin tai hapen määrää veressä. BAN yhdistää nämä sensorit langattoman yhteyden avulla keskukseen, joka puolestaan yhdistää ne eräänlaiseen valvontajärjestelmään. Kun tähän yhdistetään mobiililaitteen langaton käyttöliittymä, on mahdollista seurata, yhdistää ja keskittää tietoa. Tätä tietoa voidaan tiivistää ja tarjota terveydenhuoltopalveluiden tarjoajille esimerkiksi diagnoosin tekemisen tueksi. Pienillä laajennuksilla tällaisen teknologian avulla voidaan tuottaa esimerkiksi kehotuksia lääkkeiden ottamiseen tai terveellisiin elämäntapoihin liittyen. [Patrick *et al.* 2008.]

Myös laitteiden kontekstitietoisuus lisääntyy. Kontekstitietoisuudella tarkoitetaan sitä, että mobiilijärjestelmä tai mobiililaite kykenee tunnistamaan fyysisen ympäristönsä ja toimimaan sen mukaisesti [Ajana El Khaddar *et al.* 2012]. Mobiililaitteiden kontekstitietoisuus langattomaan teknologiaan yhdistettynä tarjoaa terveydenhuollolle paljon uusia mahdollisuuksia [Patrick *et al.* 2008]. Mobiiliviestinnän ja kontekstitietoisten mobiilipalvelujen lähentyminen tukee päätöksentekoa haastavissa ympäristöissä, jolloin lääkärit, sairaanhoitajat ja muut ammattilaiset tarvitsevat kriittistä lääketieteellistä tietoa välittömästi. [Ajana El Khaddar *et al.* 2012.]

3.4. Kritiikkiä

Mobiililaitteiden käytön yleistyminen on nostanut esiin huolen siitä, voiko niiden käytöstä olla haittaa terveydelle tai elämänlaadulle [Patrick *et al.* 2008]. Langattomien signaalien lähettäjien käytössä tulee ottaa huomioon niiden mahdolliset haittavaikutukset. Suositukset vaihtelevat käyttötaajuuksien mukaan, mutta ainakin lähettimen teho ja sen etäisyys kehosta tulisi harkita tarkkaan. Esimerkiksi kämmentietokoneet toimivat pienellä teholla, mutta niitä käytetään hyvin lähellä kehoa, kun taas esimerkiksi ambulanssien radiot toimivat suuremmilla

tehoilla, mutta niiden lähettävät antennit sijaitsevat kaukana käyttäjistä. [Ajana El Khaddar *et al.* 2012.]

Asiantuntijoiden mukaan vapaasti ladattavissa olevien terveyssovellusten takana ei myöskään välttämättä ole minkäänlaista tieteellistä näyttöä, jonka vuoksi niistä saattaa pahimmassa tapauksessa olla käyttäjälleen enemmän haittaa kuin hyötyä [Kauppalehti 2015]. Samojen sovellusten on myös uutisoitu jakavan käyttäjien tietoja kolmansille osapuolille, esimerkiksi mainos- ja analytiikkayrityksille [Taloussanomat 2013].

Huoli yksityisyydestä voi rajoittaa terveyssovellusten käytöstä saatavaa hyötyä, jos käyttäjät eivät saa hallita tiedon keräämistä ja jakamista. Ihmiset jakavat tietoa terveydestään eri tavoin yleisöstä riippuen. Jotakin tietoa saatetaan haluta jakaa mieluummin tuntemattomille kuin ystäville. Hyvällä suunnittelulla voi kuitenkin hälventää käyttäjien huolta jaetusta tiedosta. [Prasad *et al.* 2012.]

Euroopan Unionin alueiden komitea [2014] painottaa lausunnossaan, että terveysalan mobiilisovellusten kehittämisen onnistuminen on kiinni siitä, kuinka yksityisyyden suoja onnistutaan turvaamaan. Kansalaisten luottamus terveydenhuoltoon ja terveydenhuollon tiedonhallintaan on edellytys hyvälle ja turvalliselle hoidolle. Terveyssovellusten turvallisuus on siis erittäin tärkeä ja laaja alue. Niin terveydenhuollon langattomat palvelut, kuin mobiililaitteetkin, ovat jatkuvan tutkimuksen kohteena, sillä esimerkiksi tunnistautumiseen, luottamuksellisuuteen ja taltioidun tiedon suojaukseen liittyy paljon ongelmia. [Woodbridge *et al.* 2009.]

4. Terveyssovellusten käyttäjäkokemus

Terveyssovellukset voivat edistää hyvinvointia ja tukea käyttäytymismuutoksia, jos ne ovat yksinkertaisia, kiinnostavia ja sopivia jokapäiväiseen elämään. Sovellussuunnittelun tulisi tukea pieniä päivittäisiä välitöntä hyötyä aikaansaavia tekoja, painottaa itsensä kehittämistä ja pohdintaa sekä opastaa valinnanvapautta rajoittamatta. [Kaipainen 2014.] Käytettävyys ja käyttäjäkokemus ovat näissä sovelluksissa avainasemassa, sillä niitä käyttävät myös ihmiset, joilla ei ole juurikaan kokemusta mobiililaitteiden käytöstä tai teknologiasta ylipäätään [Zapata *et al.* 2015].

Mobiililaitteet tarjoavat uusia interaktiivisia mahdollisuuksia sovelluskehitykseen. Nämä mahdollisuudet ovat etenkin terveydenhuollossa erittäin hyödyllisiä, minkä vuoksi ei olekaan yllättävää, että mobiililaitteet ovat kasvattaneet suosiotaan terveyssovelluksien kohdealustana. Terveysmobiilisovelluksia käytetään yhä useammin terveydenhuollon apuvälineenä [Liu *et al.* 2011.]

Terveyssovellus tähtää käyttäjänsä auttamiseen, tukemiseen ja motivointiin, mikä onnistuu vain, jos sovellus on suunniteltu hyvin. Terveyssovelluksen

suunnittelu pitkäaikaiseen käyttöön on kuitenkin haastavaa, sillä sovellusten tulee sisältää muun muassa kannustavia elementtejä. [Ahtinen *et al.* 2009.] Myös mobiililaitteen pieni käyttöliittymä asettaa omat haasteensa sovellusten suunnittelulle.

Terveyssovellukset ovat tuore ilmiö, joten siitä, mikä toimii ja mikä ei toimi, on tehty suhteellisen vähän tutkimuksia. Jotta tällainen ymmärrys voisi kehittyä, on ensin tunnistettava terveyssovellusten tärkeimmät elementit. Terveyssovelluspalvelu tarvitsee toimiakseen kolme elementtiä: mobiililaitteen, sovellusalustan ja tietysti itse sovelluksen. Älypuhelimet ja tabletit valitaan usein terveyssovellusten alustoiksi niiden kannettavuuden tai koon vuoksi. [Liu *et al.* 2009.]

Tällä hetkellä neljä suurinta mobiilialustojen tarjoajaa ovat Google (Android), Apple (iOS), Microsoft (Windows mobile OS family) ja RIM (Blackberry) [IDC 2015].

4.1. Terveyssovellusten käyttäjäkokemuksen kehittäminen

Monet ovat jo innostuneet terveyssovelluksista ja käyttävät niitä saumattomasti osana päivittäistä elämäänsä. He tarkkailevat ja arvioivat esimerkiksi ruokavalioitaan ja liikkumistaan päivittäin. Jos esimerkiksi liikunta uhkaa jäädä liian vähälle, kehottaa sovellus käyttäjäänsä lähtemään liikkeelle erilaisin viestein ja muistutuksin. Suunnittelijat rakentavat siis järjestelmiä, jotka kaikessa hiljaisuudessa tarkkailevat ja tallentavat toimintaamme. Kehitteillä on myös jatkuvasti uusia välineitä hyvinvoinnin ja terveyden mittaamiseen. Tulevaisuudessa voikin olla mahdollista mitata esimerkiksi stressitaso ja sen vaikutukset jokapäiväiseen elämään suoraan mobiililaitteesta [Adams *et al.* 2014]. Mutta miten terveyssovelluksen käytöstä saataisiin mahdollisimman hyvä kokemus?

Terveyssovelluksilla on omat erikoispiirteensä, joiden vuoksi niiden suunnitteluun ja kehitykseen tulee ottaa hiukan erilainen lähestymistapa kuin selainpohjaisia sovelluksia suunniteltaessa. Kuten edellä mainittiin, mobiililaitteet ovat terveyssovelluksille otollisimpia laitteita. Ne tunnistavat useita kosketuspisteitä samanaikaisesti ja pystyvät uuden teknologian ansiosta tunnistamaan käyttökontekstinsa.

Terveyssovellusten kehittäjien onkin tärkeää pyrkiä innovoimaan uusia tapoja tukea näitä laitteiden vahvoja ominaisuuksia. Vaikka mobiililaitteiden kanssa voi käyttää erilaisia ulkoisia antureita, yhteensopivien anturien joukossa ei ole monia lääketieteellisiin tarkoituksiin sopivia. Terveyssovellusten ja anturien kehittäjien tulisi toimia yhdessä yhteensopivuuden parantamiseksi. [Liu *et al.* 2009.]

Kaipaisen [2014] käyttäjätutkimuksien mukaan sovelluksen pilkkominen pienempiin osiin ja konkreettisiin harjoituksiin johtaa parempaan käyttäjäkokemukseen ja osallistumiseen sovellusalustasta riippumatta. Valinnanvapauden säilyttäminen sovelluksessa saattaa johtaa parempaan lopputulokseen kuin pakottaminen, etenkin ennaltaehkäisevissä ohjelmissa. Psykologiset teoriat tulisi myös ottaa huomioon ja suunnitteluprosessissa tulisi pyrkiä käyttäjäkeskeiseen ja osallistavaan kehitykseen. [Kaipainen 2014.]

Fyysiseen aktiivisuuteen kannustavien sovellusten suunnittelussa tulisi huomioida ainakin seuraavat neljä vaatimusta [Consolvo *et al.* 2006]:

1) Anna käyttäjälle tunnusta suorituksista
2) Anna käyttäjälle palautetta etenemisestä
3) Tue sosiaalista vuorovaikutusta
4) Huomioi käyttäjän henkilökohtaiset rajoitteet.

4.1.1. Tunnustus

Sovellussuunnittelussa tulisi välttää harhauttavaa informaatiota. Esimerkiksi askelmittariin perustuvan aktiivisuusmittarin tulisi pystyä tuottamaan informaatiota myös pyöräilyretkeltä. Käyttäjien tulisi voida myös täydentää sovelluksen keräämää dataa. [Consolvo *et al.* 2006.] Tämän voi toteuttaa esimerkiksi antamalla käyttäjälle mahdollisuuden lisätä omia kommentteja informaation joukkoon, jolloin informaatiosta tulee entistä informatiivisempaa. Jotta olisi mahdollista antaa käyttäjälle paras mahdollinen tunnustus, tulee suunnittelijoiden ymmärtää kohderyhmän tarpeet, mittauslaitteiden rajoitukset ja tukea informaation täydentämisen mahdollisuus. [Consolvo *et al.* 2006.]

4.1.2. Palaute

Tutkimuksissa on havaittu ainakin kolme henkilökohtaisen edistymisen mittaria, joista tulisi tuottaa käyttäjälle palautetta: historia, nykytilanne ja aktiivisuustaso [Consolvo *et al.* 2006]. Käyttäjän motivaatioon vaikuttaa myös se, kuinka pitkältä aikaväliltä palautetta voidaan antaa. On havaittu, että sovellukset, jotka pystyvät tarjoamaan palautetta edistymisen eri osa-alueista pitkällä aikavälillä, motivoivat käyttäjiään entistä enemmän. Näin käyttäjä voi seurata edistymistään ja tiedostaa jotain tapahtuvan jo ennen kuin konkreettisia tuloksia on havaittavissa. [Ahtinen *et al.* 2009.]

4.1.3. Sosiaalisen vuorovaikutuksen tukeminen

Informaatioteknologia tehostaa myös sosiaalista vuorovaikutusta, yhteistyötä ja tiedon jakamista erilaisten tukiryhmien tai verkkosivujen muodossa. Vaikka vertaisverkon terveysvaikutukset ovat ristiriitaisia, on kuitenkin kohtuullista olet-

taa, että mobiililaitteiden luontaiset ominaisuudet edistävät esimerkiksi säännöllisempää yhteydenpitoa. [Patrick *et al.* 2008.] Sosiaalinen vuorovaikutus voidaan jakaa karkeasti kolmeen osaan; sosiaaliseen paineeseen, sosiaaliseen tukeen ja sosiaaliseen kommunikointiin [Consolvo *et al.* 2006]. Etenkin fyysiseen aktiivisuuteen kannustavissa sovelluksissa aktiivisuustason ja edistymisen jakaminen ystävien kanssa luo paineen, jonka ansiosta käyttäjä saavuttaa omat tavoitteensa helpommin. Samaan tapaan sosiaalinen tuki voi kannustaa saavuttamaan omat tavoitteet. Usein tueksi riittää vain tieto siitä, mitä muille käyttäjille kuuluu, mutta myös kommunikoinnin mahdollisuutta tulisi tukea, sillä se antaa käyttäjille vapauden esimerkiksi kuvailla tekemisiään. [Consolvo *et al.* 2006.]

4.1.4. Henkilökohtaisten rajoituksien huomioiminen

Kun suunnitellaan sovelluksia päivittäisen aktiivisuuden mittaamiseen, tulisi ottaa huomioon useita käytännön asioita. Sovelluksen toimintojen tulisi olla mahdollisimman integroituja ja helppokäyttöisiä, mutta myös käytettävien laitteiden tarpeellisuutta ja kokoa tulisi miettiä. Suunnittelussa tulisi pyrkiä käyttämään mahdollisimman vähän uusia puettavia laitteita, sillä niiden käyttö ei aina ole miellyttävää. Jos niitä kuitenkin täytyy käyttää, tulisi niiden ulkomuoto harkita hyvin tarkkaan. [Consolvo *et al.* 2006.]

4.2. Tulevaisuus

On siis selvää, että terveyssovellukset voivat parantaa merkittävästi terveydenhuollon ja terveyskasvatuksen tehokkuutta ja laatua. Terveyssovellusten suunnittelussa on kuitenkin paljon kehitettävää, jotta täysi hyöty mobiilialustan ominaisuuksista saataisiin käyttöön. Erityisesti kaksi- tai kolmiulotteinen visualisointi sekä käyttökontekstin ja ulkoisten anturien syvempi integrointi voisivat edistää terveyssovellusten käytettävyyttä ja hyödyllisyyttä. [Liu *et al.* 2009.] Interaktiivinen lähestymistapa taivuttelee käyttäjää kontekstiin tai aikaan sopivan palautteen ja ohjeistuksen avulla [Ahtinen *et al.* 2009] ja juuri sovelluksen interaktiivisuuteen tulisi tässä konseptissa kiinnittää erityistä huomiota.

5. Mobiilisovellusten käyttäjäkokemuksen arviointimenetelmät

Käytettävyys ja käyttäjäkokemus kulkevat käsi kädessä. Käsitteiden välinen raja on häilyvä ja niitä on jopa yritetty täysin erottaa toisistaan. Yleisesti voidaan kuitenkin ajatella, että käytettävyys sisältyy käyttäjäkokemukseen. Siksi käyttäjäkokemuksen arviointi on pitkälti jo olemassa olevien käytettävyyden arvioinnin menetelmien täydentämistä. [Vermeeren *et al.* 2010.] Onnistunut tuotekehitys vaatii syvällistä ymmärrystä käyttäjien toimista, tyyleistä ja haluista. Käyttäjien

tarpeiden ymmärtäminen on merkittävä, mutta samalla usein myös tuotekehityksen heikoin lenkki. Tuotteen käyttöä koskevat tieto on keskeistä niin tuotteen tekniselle toteutukselle, markkinoinnille, liiketoiminnalle, teknisen tuen suunnittelulle kuin tietysti itse käyttäjillekin. Usein käyttäjätutkimukseen investointi palautuu jo tuotteen teknisen toteutuksen aikana. [Hyysalo 2009.] Hyysalon [2009] mukaan onnistunut teknologia on haluttava, hyödyllinen, käytettävä ja miellyttävä sekä sisältää mahdollisimman vähän kaikkea sellaista, joka häiritsee näiden ominaisuuksien toteutumista.

Käytettävyystestit keskittyvät usein vain tehtävien suorittamisen arviointiin, kun käyttäjäkokemus taas keskittyy enemmän elettyihin kokemuksiin. Käyttäjäkokemus on subjektiivinen käsite, joten haluamme tietää, mitä käyttäjä tuntee käyttäessään tuotetta. Vaikka pelkässä käytettävyyden arvioinnissa on monia käyttäjäkokemuksen arvioinnille hyödyllisiä komponentteja (esimerkiksi käyttäjän tyytyväisyys), käyttäjäkokemus laajentaa näkemystä moniin muihinkin subjektiivisiin ominaisuuksiin. Käyttäjän motivaatiolla ja odotuksilla on vahvempi rooli käyttäjäkokemuksen arvioinnissa kuin perinteisessä käytettävyyden arvioinnissa. [Vermeeren *et al.* 2010.]

Maguiren [2001] mukaan käytettävyyden ja käyttäjäkokemuksen arvioinnille on kaksi syytä: ensinnäkin tuotetta voidaan parantaa arvioinnilla kehitysprosessin aikana tunnistamalla ja korjaamalla käytettävyysongelmia ja toiseksi on tärkeää selvittää, voivatko käyttäjät käyttää tuotetta onnistuneesti. Suunnitteluratkaisuja tulisi arvioida läpi tuotekehitysprosessin esimerkiksi paperiprototyyppejä käyttäen.

Hyvän käyttäjäkokemuksen saavuttaminen vaatii seuraavien vaiheiden toteutumista [Maguire 2001]:

1) Käyttäjäkeskeisen suunnitteluprosessin huolellinen suunnittelu
2) Käyttökontekstin ymmärtäminen vaatimusten ja sovelluksen arvioinnin perustana
3) Käyttäjävaatimusten ymmärtäminen tavalla, joka on saavutettavissa
4) Sovellus- ja käyttöliittymäsuunnittelu iteratiivisesti ja joustavasti
5) Käytettävyyden arviointi sekä asiantuntijoiden, että käyttäjien näkökulmasta sopivissa vaiheissa projektia.

Erilaisia menetelmiä käytettävyyden ja käyttäjäkokemuksen arviointiin on monia. Maguire [2001] jakaa menetelmät karkeasti osallistaviin, avustettuihin ja valvottuihin menetelmiin. Menetelmät voidaan luokitella myös esimerkiksi suunnitteluvaiheen tai käytettävissä olevan ajan mukaan [Rajeshkumar *et al.* 2013]. Käyttäjätestauksen sijainti vaikuttaa myös menetelmän valintaan. Sijaintina voi olla esimerkiksi laboratorio, kenttä tai verkko. [Vermeeren *et al.* 2010.]

Menetelmien luokitteluun on siis olemassa monia erilaisia tapoja. Jotta yleisesti hyväksyttävä näkemys käyttäjien kokemusten arviointiin voitaisiin määritellä, on menetelmistä tehtävä lisätutkimuksia [Vermeeren *et al.* 2010].

5.1. Osallistavat menetelmät

Osallistavassa arviointimenetelmässä käyttäjät käyttävät prototyyppiä tehtävien suorittamisessa. Menetelmässä käytetään usein ”ääneen ajattelu” -nimistä tekniikkaa, jolloin käyttäjää pyydetään puhumaan ääneen kaikki tehtävien suorittamisen aikana tulleet ajatukset. Testitapahtuma äänitetään tai videoidaan myöhempää analyysiä varten. [Maguire 2001.]

Arviointityöpaja on edellä esitetyn perinteisen menetelmän muunnos. Siinä käyttäjät yrittävät tehdä tehtäviä kehittäjien tarkkaillessa heitä. Testaustilaisuuden jälkeen kehittäjät ja käyttäjät keskustelevat esiin nousseista ongelmista. Menetelmä sopii monien erilaisten sovellusten testaamiseen ja sen vahvuutena voidaan pitää sitä, että se tuo kehittäjät ja käyttäjät yhteen ohjatussa ympäristössä. Monen käyttäjän samanaikainen testaus tuo myös esiin saman ongelman eri puolia. [Maguire 2001.]

Läpikävely on myös osallistavan arviointimenetelmän muunnos. Se on prosessi, jossa tuotteen kannalta relevantit henkilöt käyvät tuotteen läpi askel askeleelta ennalta määrättyjen tehtävien avulla. Käytettävyysasiantuntija arvioi käyttäjän edistymistä ja listaa lopuksi ongelmien vakavuuden. [Maguire 2001.]

5.2. Avustetut menetelmät

Avustetuissa arviointimenetelmissä käyttäjää pyydetään tekemään tehtävien sarja tarkkailijan läsnä ollessa. Käyttäjää pyydetään suorittamaan tehtävät ilman apua, mutta tarkkailija voi avustaa käyttäjää tämän jäädessä jumiin. Tämä menetelmä kertoo etenkin sen, kuinka hyvin järjestelmä tukee tehtävien suorittamista, mutta antaa myös käyttäjälle mahdollisuuden antaa palautetta testauksen edetessä. [Maguire 2001.]

5.3. Valvotut menetelmät

Valvottu käyttäjätestaus on kaikkein antoisin tutkimusmenetelmä, jossa kohdekäyttäjät testaavat prototyyppiä valvotussa tilassa. Menetelmä antaa suoran vastauksen siihen, miten käyttäjät käyttävät tuotetta ja mitkä ominaisuudet aiheuttavat ongelmia. Tapoja informaation keräämiseen on useita, testitilanteet voidaan esimerkiksi videoida. [Maguire 2001.]

5.4. Heuristinen arviointi

Heuristisessa arvioinnissa yksi tai useampi asiantuntija arvioi prototyypin yrittäen tunnistaa mahdollisia kompastuskiviä. Menetelmään suositellaan käytettävän useampia asiantuntijoita, jotta henkilökohtaisten ennakkoluulojen ja kokemuksien vaikutukset saadaan minimoitua. Tämän menetelmän avulla saadaan palautetta ja suosituksia nopeasti, mutta asiantuntijan voi olla vaikea omaksua käyttäjän rooli. [Maguire 2001.] Heuristinen arviointi on arvioinnin epävirallisin muoto [Nielsen 1995].

5.5. SMASH: heuristiikat älypuhelimille

Älypuhelin- ja mobiililaitemarkkinoilla on kova kilpailu. Käyttäjät keskittyvät muun muassa laitteen visuaalisuuteen, ergonomiaan, suorituskykyyn ja käyttäjäkokemukseen. Parhaan tuloksen varmistamiseksi tulee tuotteen arvioinnissa käyttää oikeita heuristiikkoja, siksi uusia menetelmiä kehitetään jatkuvasti. SMASH on tuore erityisesti älypuhelimille ja mobiilisovelluksille kehitetty kahdentoista käytettävyysheuristiikan joukko, jonka avulla asiantuntijat voivat arvioida mobiililaitteiden käyttäjäkokemusta aiempaa heuristista arviointia tehokkaammin. Heuristiikkojen hyödyllisyys ja tehokkuus on kokeellisesti vahvistettu. [Inostroza *et al.* 2016.]

5.6. Muut menetelmät

Käyttäjäkokemusta voidaan arvioida monia muitakin menetelmiä apuna käyttäen. Muun muassa tyytyväisyyskysely, kognitiivisen työmäärän mittaaminen tai kriittisten tilanteiden kartoittaminen voivat olla hyviä tapoja kartoittaa käyttäjien kokemuksia. [Maguire 2001.]

Tyytyväisyyskysely pyrkii selvittämään käytön aikana muodostuneet vaikutelmat. Menetelmässä käyttäjät täyttävät kyselylomakkeen sovelluksen käytön jälkeen. Kognitiivisen työmäärän arvioinnilla taas tarkoitetaan sitä, kuinka suuren henkisen työn käyttäjä joutuu tekemään käyttäessään sovellusta. Tässä menetelmässä voidaan kerätä tietoa esimerkiksi käyttäjän sykkeestä. Kriittisillä tilanteilla puolestaan tarkoitetaan ongelmia, joissa suunnittelussa ilmenee huomattavia puutteita. Menetelmässä käyttäjä antaa suullisen raportin tapahtumasta, joka myöhemmin analysoidaan ja sen vakavuus määritellään. Menetelmä voi olla hyvin ekonominen tapa kerätä tietoa, mutta ei välttämättä luotettavin, sillä se riippuu käyttäjän muistista. Haastattelut ovat myös nopea ja edullinen tapa kerätä palautetta käyttäjiltä, mutta niitä harvoin käytetään ainoana arviointimenetelmänä. Haastatteluja käytetään usein viimeisenä osana käyttäjätestausta. [Maguire 2001.]

6. Case: MeeDoc – heuristinen arviointi SMASH heuristiikkojen avulla

MeeDoc on mobiilisovellus, joka tuo lääkärin kenen tahansa ulottuville paikasta riippumatta [Qvik 2013]. Pääsy laadukkaaseen terveydenhuoltoon on maailmanlaajuisesti kasvava ongelma, johon MeeDoc pyrkii vastaamaan. MeeDocin avulla käyttäjä voi kommunikoida lääkärin kanssa missä tahansa, sillä lääkäri arvioi hoidontarpeen videokuvan avulla. Kuluttajien ja yritysten työterveyden lisäksi MeeDoc auttaa myös terveydenhuollon ammattilaisia tarjoamalla mahdollisuuden esimerkiksi etälääkärivastaanottoihin. Palvelu välittää myös reseptit suoraan apteekkiin ja se on käytössä jo muun muassa Ylioppilaiden Terveydenhoitosäätiöllä. [Qvik 2013.]

MeeDoc voitti Grand One 2014 -kilpailussa pääpalkinnon sekä parhaan kuluttajille suunnatun palvelun kategorian. Samana vuonna MeeDoc palkittiin myös Best Mobile Service in Finland -kilpailussa parhaana mobiilisovelluksena eHealth service -kategoriassa. [Qvik 2013.]

Seuraavaksi suoritan lyhyen heuristisen arvioinnin MeeDoc-sovellukselle edellä esittelemieni SMASH-heuristiikkojen [Inostroza *et al.* 2016] avulla. Laitteena käytössäni on Apple iPhone 6, joten ohjelmistona toimii iOS 9.2. Heuristiikkoja on kaksitoista: tuotteen tilan näkyvyys, tuotteen ja tosielämän vastaavuus, käyttäjän kontrolli ja vapaus, tuotteen johdonmukaisuus, virheiden ehkäisy, käyttäjän muistikuorman minimointi, kustomointi ja oikotiet, käytön tehokkuus, esteettinen suunnittelu, virheiden tunnistaminen ja niistä toipuminen, ohjeet ja dokumentointi sekä fyysinen vuorovaikutus ja ergonomia [Inostroza *et al.* 2016].

6.1. Arviointi

MeeDoc on ladattavissa Applen App Storesta maksuttomasti. Alussa käyttäjälle esitellään palvelu lyhyesti. Käyttäjä voi pyyhkäisemällä vasemmalle lukea esittelysivuja, mikä tukee luonnollista dialogia. Käyttäjä kuitenkin toivotetaan tervetulleeksi vasta viimeisellä sivulla, mikä puolestaan ei tue luonnollisen järjestyksen heuristiikkaa.

Alussa käyttäjälle annetaan kaksi vaihtoehtoa: rekisteröityminen tai kirjautuminen. Rekisteröinnin tai kirjautumisen valittuaan käyttäjä ei kuitenkaan pääse aloitussivulle takaisin mistään, mikä saattaa hämmentää käyttäjää. Tämä myös rikkoo hallitsemisen ja vapaudentunteen heuristiikkaa.

Rekisteröityminen onnistuu sovelluksessa helposti. Käyttäjän tulee vain kirjoittaa sähköposti, salasana ja hyväksyä palvelun käyttöehdot. Sähköpostia tai

salasanaa ei kuitenkaan tarvitse vahvistaa tai toistaa mitenkään, jolloin esimerkiksi kirjoitusvirhe estäisi seuraavan kirjautumisen. Tämä rikkoo virheiden ehkäisyn heuristiikkaa.

Sovelluksen käyttöliittymä on yksinkertainen. Aloitusnäkymässä voi tutustua MeeDoc-palveluun tarkemmin tai aloittaa keskustelun lääkärin kanssa. Alareunassa on kuvakkeet, joista voi navigoida muun muassa oman tilin hallintaan. Aktiivinen kuvake on väriltään sininen, muiden kuvakkeiden ollessa harmaita, jolloin palvelun tila on helppo havaita, eikä käyttäjän muisti kuormitu turhaan.

Kaikki tarvittava ohjeistus on helposti saatavilla. Jokainen sivu on myös nimetty informatiivisesti. Yhteneväisyyden ja ohjeistuksen heuristiikat ovat siis tuettuina.

Käyttäjä voi halutessaan luoda sovellukseen lyhemmän PIN-koodin, mikä tekee kirjautumisesta jatkossa helpompaa. Tämä ominaisuus tukee käytön tehokkuuden heuristiikkaa. PIN-koodi helpottaa myös sovelluksen sisällä navigoimista, sillä esimerkiksi lääkärin ohjeiden ja oman tilin tarkastelu vaativat aina salasanan uudestaan. Sovellus ei tue Applen sormenjälkitunnistusta, mikä olisi salasanaa turvallisempi tilinsuojaustekniikka, sillä se on vaikeammin kopioitavissa.

6.2. Pohdinta

SMASH-heuristiikkoihin verraten MeeDoc-sovellus tarjoaa miellyttävän käyttäjäkokemuksen. Sovelluksen tila on aina havainnollistettu, dialogi etenee loogisesti, metaforat vastaavat tosielämää, design on minimaalinen sekä ohjeet ja dokumentointi on selkeästi saatavilla. En myöskään havainnut fyysisen vuorovaikutuksen esteitä, tosin pitkässä videopuhelussa lääkärin kanssa voi ongelmaksi muodostua puhelimen sijoittelu. Sijainnilla on tässä palvelussa suuri merkitys myös yksityisyyden suojan kannalta. Konsultaatio ei onnistu viestien välityksellä, joten konsultaatioon täytyy tapahtua rauhallisessa ja yksityisessä paikassa. Rekisteröitymisvaiheessa sähköpostiosoite ja salasana tulisi kuitenkin varmistaa virheiden ehkäisemiseksi. Tämä saattaa olla vakava käytettävyysongelma, sillä se voi estää uuden kirjautumisen kokonaan. Kustomointimahdollisuuksia sovellus ei tarjoa.

Kaiken kaikkiaan MeeDoc-sovellus on heuristiikkojen varjossa käyttäjäkokemukseltaan hyvä etälääkäripalvelu, jolle on varmasti kysyntää myös tulevaisuudessa.

7. Yhteenveto

Terveydenhuollon resursseista on pulaa ja palvelurakenneuudistusten myötä matka palveluntarjoajan luo saattaa kasvaa hyvinkin pitkäksi. Samaan aikaan

monet terveysongelmat lisääntyvät, joten tehokkaiden terveysinterventioiden tekemiseen tulisikin löytää kaava mahdollisimman nopeasti. Terveyssovelluksilla on monia erikoispiirteitä, joiden vuoksi niiden suunnittelu ja arviointi on erilaista muihin mobiilisovelluksiin verrattuna. Sovellusten suunnittelussa tulee muun muassa kiinnittää muita sovelluksia tarkemmin huomiota yksityisyyden suojaan. Käyttäjäkokemuksella on erityisen suuri merkitys terveysinterventioiden tekemisessä, sillä se erottaa menestymättömän ja menestyvän sovelluksen toisistaan. Huono käyttäjäkokemus tarkoittaa automaattisesti huonoa sovellusta. Käyttäjäkokemus on laaja ja suhteellisen tuore käsite, joka usein sekoitetaan pelkkään käytettävyyteen.

On selvää, että terveyssovellukset voivat parantaa merkittävästi terveydenhuollon ja terveyskasvatuksen tehokkuutta ja laatua. Terveyssovellusten suunnittelussa on kuitenkin paljon kehitettävää, jotta täysi hyöty mobiilialustan ominaisuuksista saataisiin käyttöön. Erityisesti kaksi- tai kolmiulotteinen visualisointi sekä käyttökontekstin ja ulkoisten anturien syvempi integrointi voisivat edistää terveyssovellusten käytettävyyttä ja hyödyllisyyttä. Sekä terveyssovelluksista, että käyttäjäkokemuksesta tarvitaan kuitenkin lisää tutkimuksia, jotta sovellusten käyttäjäkokemukselle ja sen arvioinnille voitaisiin määrittää selkeä kehys.

Viiteluettelo

Phil Adams, Eric P.S. Baumer and Geri Gay. 2014. Staccato social support in mobile health applications. In: *Proc of the SIGCHI Conference on Human Factors in Computing Systems*, 653-662.

Aino Ahtinen, Elina Mattila, Antti Vaatanen, Lotta Hynninen, Jukka Salminen, Esa Koskinen and Klaus Laine. 2009. User experiences of mobile wellness applications in health promotion: User study of Wellness Diary, Mobile Coach and SelfRelax. In: *Proc. of the 3rd International Pervasive Computing Technologies for Healthcare*, 1-8.

Sunny Consolvo, Katherine Everitt, Ian Smith and James A. Landay. 2006. Design Requirements for Technologies that Encourage Physical Activity. In: *Proc. of the SIGCHI Conference on Human Factors in Computing Systems*, 457-466.

Mehdia Ajana El Khaddar, Hamid Harroud, Mohammed Boulmalf, Mohammed Elkoutbi and Ahmed Habbani. Emerging wireless technologies in e-health: Trends, challenges, and framework design issues. 2012. In: *Proc. of the International Conference on Multimedia Computing and Systems*, 440 – 445.

Esteban Angulo and Xavier Ferre. 2014. A case study on cross-platform development frameworks for mobile applications and UX. In: *Proc. of the 15th International Conference on Human Computer Interaction*, Article 27.

Euroopan Unioni. 2014. Terveysalan mobiilisovellukset. Lausunto. 109. täysistunto. Ladattu 14.12.2015.

European Commission: Digital Agenda for Europe. 2015. MHealth. https://ec.europa.eu/digital-agenda/en/mhealth. Checked 17.11.2015

FDA. 2015. Mobile Medical Applications. http://www.fda.gov/MedicalDevices/DigitalHealth/MobileMedicalApplications/default.htm#a. Checked 10.11.2015.

Sampsa Hyysalo. 2009. Käyttäjä tuotekehityksessä. Tieto, tutkimus ja menetelmät. Taideteollinen korkeakoulu.

IDC. 2015. Smartphone OS Market Share, 2015 Q2. http://www.idc.com/prodserv/smartphone-os-market-share.jsp. Checked 14.12.2015.

Rodolfo Inostroza, Cristian Rusu, Silvana Roncagliolo, Virginica Rusu and César A. Collazos. 2016. Developing SMASH: A set of SMArtphone's uSability Heuristics. *Computer Standards & Interfaces* 43, 40-52.

ISO DIS 9241-210. 2010. Ergonomics of human system interaction - part 210: Human-centred design for interactive systems. Tech. rep., International Organization for Standardization.

Kirsikka Kaipainen. 2014. *Design and Evaluation of Online and Mobile Applications for Stress Management and Healthy Eating*. Doctoral thesis. Faculty of Computing and Electrical Engineering, Tampere University of Technology.

Kauppalehti. 2015. Humpuukia ja ahdistusta – Lääkärit tyrmäävät terveyssovellukset. http://www.kauppalehti.fi/uutiset/humpuukia-ja-ahdistusta–laakarit-tyrmaavat-terveyssovellukset/L4rtyYtT. Viitattu 11.12.2015.

Predrag Klasnja and Wanda Pratt. 2012. Healthcare in the pocket: Mapping the space of mobile-phone health interventions. *Journal of Biomedical Informatics* 45, 1, 184-198.

Basant Kumar, S. P. Singh and Anand Mohan. 2010. Emerging Mobile Communication Technologies for Health. In: *Proc. of the International Conference on Computer and Communication Technology*, 828 - 832.

Kati Kuusinen and Tommi Mikkonen. 2014. On designing UX for mobile enterprise apps. In: *Proc. of the 40th Euromicro Conference on Software Engineering and Advanced Applications*, 221-228.

Chang Liu, Qing Zhu, Kenneth A. Holroyd and Elizabeth K. Seng. 2011. Status and trends of mobile-health applications for iOS devices: a developer's perspective. *Journal of Systems and Software* 84, 11, 2022-2033.

Martin Maguire. 2001. *Methods to support human-centred design*. HUSAT Research Institute. Loughborough University. Available online at http://www.idealibrary.com.

Alexander G. Mirnig, Alexander Meschtscherjakov, Daniela Wurhofer, Thomas Meneweger, Manfred Tscheligi. 2015. A Formal analysis of the ISO 9241-210 definition of user experience. In: *Proc. of the 33rd Annual Human Factors in Computing Systems*, 437- 450.

Abdul Hakim H. M. Mohamed, Hissam Tawfik, Dhiya Al-Jumeily and Lin Norton. 2011. MoHTAM: A technology acceptance model for mobile health applications. In: *Developments in E-systems Engineering*, 13-18.

Yelena Nakhimovsky, Dean Eckles and Jens Riegelsberger. 2009. Mobile user experience research: challenges, methods & tools. In: *Human Factors in Computing Systems*, 4795-4798.

Jacob Nielsen. 1993. *Usability Engineering*. Academic Press.

Jacob Nielsen. 1995. Usability inspection methods. In: *Proc. of the CHI 95 Conference Companion on Human Factors in Computing Systems*, 377-378.

Kevin Patrick, William G. Griswold, Fred Raab and Stephen S. Intille. 2008. Health and the Mobile Phone. *American Journal of Preventive Medicine* 35, 2, 177-181.

Aarathi Prasad, Jacob Sorber, Timothy Stablein, Denise Anthony, David Kotz. 2012. Understanding sharing preferences and behavior for mHealth devices. In: *Proceedings of the 2012 ACM Workshop on Privacy in the Electronic Society*, 117-128.

Qvik. 2013. Meedoc. http://qvik.fi/portfolios/meedoc-2/. Viitattu 1.12.2015.

S. Rajeshkumar, Ridha Omar and Murni Mahmud. 2013. Taxonomies of User Experience (UX) Evaluation Methods. In: *Proc. of the 3rd International Conference on Research and Innovation in Information Systems,* 533-538.

Simon Robinson, Gary Marsden and Matt Jones. 2014. *There's Not an App for That: Mobile User Experience Design for Life*. Morgan Kaufmann.

Taloussanomat. 2013. Moni terveyssovellus lavertelee tietosi mainostajalle. http://www.taloussanomat.fi/ihmiset/2013/09/02/moni-terveyssovellus-lavertelee-tietosi-mainostajille/201312174/137. Viitattu 11.12.2015.

TechTarget. 2011. Health apps definition. http://searchhealthit.tech- target.com/definition/health-apps. Checked 11.11.2015.

The Statistics Portal. 2015. Number of apps available in leading app stores as of July 2015. http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/. Checked 10.11.2015.

The Statistics Portal. 2015. Number of smartphone users* worldwide from 2012 to 2018 (in billions). http://www.statista.com/statistics/330695/number-of- smartphone-users-worldwide/. Checked 1.10.2015.

Arnold P. O. S. Vermeeren, Effie Lai-Chong Law, Virpi Roto, Marianna Obrist, Jettie Hoonhout and Kaisa Väänänen-Vainio-Mattila. 2010. User experience evaluation methods: current state and development needs. In: *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries*, 521-530.

Wikipedia. 2013. Jokapaikan tietotekniikka. https://fi.wikipedia.org/wiki/Jokapaikan_tietotekniikka. Viitattu 1.12.2015.

Jonathan Woodbridge, Ani Nahapetian, Hyduke Noshadi, Majid Sarrafzadeh and William Kaiser. 2009. Wireless health and the smartphone conundrum. *ACM SIGBED Review* 6, 2. Article 11.

Yle uutiset. 2015. Soten lopputulos: Päivystäviä sairaaloita jää 12 – osa nykyisistä keskussairaaloista joutuu supistamaan toimintaansa. http://yle.fi/uutiset/soten_lopputulos_paivystavia_sairaaloita_jaa_12__osa_nykyisista_keskussairaaloista_joutuu_supistamaan_toimintaansa/8440863. Viitattu 17.11.2015.

Belén Cruz Zapata, José Luis Fernández-Alemán, Ali Idri and Ambrosio Toval. 2015. Empirical studies on usability of mHealth apps: a systematic literature review. *Journal of Medical Systems* 39, 2. 20 pages.

Katsaus valvontakamerajärjestelmien yksityisyyden suojausmenetelmiin

Anniina Seppä

Tiivistelmä.
Valvontakamerat ja erilaiset videokamerajärjestelmät tallentavat yhä suuremman osan ihmiselämästä filmille. Kamerajärjestelmien laajuus ja niiden entistä moniulotteisemmat käyttötarkoitukset ovat herättäneet keskustelua siitä, miten varmistettaisiin kuvattavien yksityisyyden suoja. Tämä katsaus pohtii yksityisyyttä ja videokameravalvontaa sekä niiden suhteita toisiinsa. Käsittelyssä on erilaisia yksityisyyssuojausmenetelmiä, joilla on pyritty vastaamaan kasvavaan huolenaiheeseen. Tällaisia ovat yksityisyyssuodattimet eli visuaaliset kuvanmuokkaustekniikat. Lisäksi tutustutaan fyysiseen mekaniikkaan perustuviin keinoihin, joilla kuvattava voi seurata tiedonkulkua ja vaikuttaa valvontakameroiden toimintaan. Ehdotetut menetelmät ovat lupaavia, mutta niillä on vielä omat ongelmansa. Ne voivat kuitenkin olla voitettavissa lisätutkimuksilla, yhteiskunnallisin sopimuksin ja kuvattavien valvontatietoisuuden lisäämisellä.

Avainsanat ja -sanonnat: valvontakamerat, yksityisyys, yksityisyyssuodattimet.

1. Johdanto

Valvontakamerat ja erilaiset videokamerajärjestelmät ovat vähitellen tulleet pysyväksi osaksi urbaania ympäristöä. Niitä käytetään niin julkisissa kuin yksityisissä tiloissa erilaisiin tarkoitusperiin: esimerkiksi valvontaan, opetukseen, terveydenhuoltoon ja viihdytykseen.

Koska kameroita on paljon ja niitä sovelletaan eri tehtävissä, on selvää, että ne myös tallentavat yhä suuremman osan ihmiselämän jokapäiväisestä arjesta videolle. Videoitu kuvamateriaali voi sisältää kuvauksen kohteelle sensitiivistä informaatiota – jota ei välttämättä edes tarvita kamerajärjestelmän alkuperäistä tehtävää varten. Pahimmassa tapauksessa tiedon paljastuminen ja leviäminen voisivat aiheuttaa vahinkoa kuvatulle henkilölle.

Valvontakamerat tunkeutuvat ihmisille herkkään yksityiselämään. Tästä huolimatta kameroiden valmistajat ja niiden operoijat eivät ole vielä ottaneet vastuutaan tosissaan. Sanomalehdissä on toisinaan juttuja esimerkiksi vauvojen itkuhälyttimistä, joista on havaittu jälkikäteen suuria tietoturva-aukkoja; toisaalta voidaan lukea uutisia siitä, kuinka kaupan valvontakameroiden kuvamateriaali on vuodettu laittomasti sosiaaliseen mediaan. Vaikka vastaavanlaiset tapaukset aiheuttavat ongelmia niin tuotteiden valmistajille kuin kuvamateriaalin

vuotajille, ei se tule korvaamaan jo aiheutettua vahinkoa: menetettyä yksityisyyttä.

Edellä mainituista syistä olen kiinnostunut erityisesti valvottavan asemasta ympäristössä, jossa käytetään valvontakameroita. Helpoin tapa yksityisyyden vaalimiseksi olisi poistaa valvontakamerat kokonaan. Tällaisen toimenpiteen myötä menetettäisiin kuitenkin myös kameroista saatavat hyödyt, minkä takia en näe kyseistä ratkaisua mielekkäänä. Sen sijaan aionkin keskittyä katsauksessani keinoihin, joilla suojattaisiin yksityisyyttä muilla keinoin ja annettaisiin lisää vaikutusvoimaa valvottavalle osapuolelle.

Pohdin toisessa luvussa yksityisyyttä, valvontaa ja kameraverkostoja sekä sitä, kuinka nämä liittyvät toinen toisiinsa. Sen jälkeen etenen tutkimaan erilaisia keinoja, joilla pyritään säilyttämään kuvattavien yksityisyyssuoja ilman, että kamerajärjestelmistä tulee tehottomia. Kolmannessa luvussa aiheena on *yksityisyyssuodattimet* (engl. privacy filter) eli visuaaliset keinot, joilla videokuva peitetään osittain tai kokonaan. Neljännessä luvussa keskitytään keinoihin, joilla kuvattavat voivat vaikuttaa siihen, miten heitä kuvataan ja kenelle. Viides luku esittelee ehdotuksia, kuinka valvottava voi seurata omien yksityistietojensa kulkua. Lopuksi pohditaan, mitä haasteita tai puutteita on nykyisillä ratkaisumalleilla.

2. Yksityisyys, valvonta ja kameraverkostot

Yksityisyydestä on vaikeaa saada yksimielistä käsitystä. Yksityisyyden tarvetaso riippuu yksilön omista, subjektiivisista tuntemuksista, jotka ovat saaneet vaikutteita ympäröivästä kulttuurista. Tämän lisäksi yksityisyyden rajat voivat olla erilaiset riippuen kontekstista. [Westin 2003.] Esimerkiksi lähimmäiselle saatetaan paljastaa paljon henkilökohtaisempia asioita kuin vieraalle; rikostutkinnassa puolestaan voidaan hyväksyä jonkinasteinen tunkeutuminen epäillyn yksityiselämään. Moniulotteisuudestaan huolimatta yksityisyys voidaan kuitenkin nähdä valtana määritellä itse, mitä ja kenelle itsestään kertoo [Westin 1967].

Valvonta vähentää tätä valtaa. Bennett ja muut [2014] määrittelevät valvonnan henkilöiden järjestelmälliseksi seuraamiseksi, joka asettaa valvottavat ja valvojat eriarvoisiin asemiin: valvojat kontrolloivat valvottavia sekä päättävät, mitä oikeuksia heille annetaan ja mitä oikeuksia heiltä puolestaan poistetaan. Valvonnan puolustajat vetoavat usein yksilön ja yhteisön yhteisen hyvän edistämiseen; vastustajat puolestaan kyseenalaistavat valvonnan todellisen tehokkuuden, tarpeen ja eettisyyden [Bennett *et al*. 2014; Schneier 2013; Kroener 2014].

Varsinkin kameravalvonta herättää keskustelua yksityisyyden rajoista. Kameraverkostot ovat laajentuneet ajan saatossa, minkä lisäksi niiden käyttötarkoitukset ovat monipuolistuneet: kamerasensoreita voidaan nykyisin käyttää paitsi

rikosten torjuntaan mutta myös viihteeseen sekä terveydenhuoltoon [Winkler and Rinner 2010]. Entistä suurempi osa ihmisten elämästä tallentuu siis videolle.

Kameroiden käytön suosio perustuu pitkälti niiden yksinkertaisuuteen: muunlainen valvontamenettely voisi joko olla mahdoton tai vaatia monien erilaisten sensoreiden soveltamista. Tämä johtuu siitä, että videon kuvamateriaali sisältää yleensä paljon enemmän informaatiota kuin pelkät numerot tai teksti. Informaation yltäkylläisyyden kääntöpuolena on toisaalta se, että kamerat tallentavat myös paljon tehtävälleen epäolennaista dataa. [Meyer and Rakotonirainy 2003.] Esimerkiksi kaupan valvontakameroiden tarkoituksena on yleensä varkaiden paikantaminen; videonauhalle tallentuu kuitenkin suurimman osan ajasta rehellisten asiakkaiden toimintaa liiketiloissa. Kamerat voivat näin ollen toteuttaa tehtävänsä, mutta kyseenalaistettavalla hinnalla.

Valvontakameroiden nykyinen kehityssuunta aiheuttaa myös uusia yksityisyys- ja tietoturvahaasteita. Kun kamerat välittävät dataa toisilleen ja keskusyksikölle langattomasti, ulkopuolisten hyökkäysten riski lisääntyy. Toisin kuin analogisia tallenteita, digitaalista dataa voidaan säilyttää suuria määriä pitkiltä aikajaksoilta, minkä lisäksi sitä on helppoa muokata, organisoida ja jakaa. [Moncrieff *et al*. 2009.] Tiedonlouhintaa ja kameroiden ihmistunnistusta voitaisiin käyttää yksittäisestä henkilöstä tehtäviin päätelmiin – esimerkiksi minkälainen persoonallisuus hänellä on [Bennett *et al*. 2014]. Yhtenä kauhunäkymänä nähdään myös niin kutsuttu Ihmis-Google, jonka avulla tietyn ihmisen sijainnin voisi löytää hetkessä.

Kuvattavien tietämättömyys kameroiden läsnäolosta ja toiminnasta koetaan ongelmalliseksi: valvonnasta varoittavat tarrat ja kyltit eivät takaa sitä, että tilassa kulkevat henkilöt antaisivat suostumuksensa heidän kuvaamisekseen. Ihmiset eivät voi esittää vaatimuksia tai toiveita sen suhteen, miten heidän yksityisyyttään vaalitaan – kuten voisiko operaattori esimerkiksi myydä tai jakaa heistä otettua videokuvaa kolmansille osapuolille. Valvottavat eivät voi myöskään varmistaa millään tavalla, että valvontakamerat seuraisivat hyviä tietoturvakäytänteitä. [Winkler and Rinner 2014.] Myös valvonnan syyt ja osapuolet voivat jäädä hämäriksi [Könings *et al*. 2013].

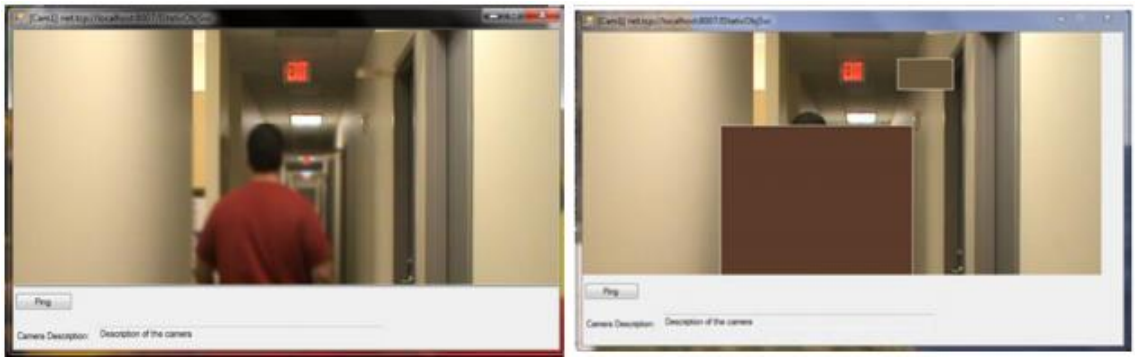
3. Yksityisyyssuodattimet

Vaikka kehittynyt tekniikka on aiheuttanut uusia yksityisyysongelmia, on sen myötä tullut myös uusia keinoja yksityisyyden turvaamiseksi, esimerkiksi *yksityisyyssuodattimet* (engl. privacy filter). Yksityisyyssuodattimet ovat joukko algoritmeja, joilla pyritään salaamaan valvottavan henkilöllisyys. Tavoitteena on *huomioalueiden* (engl. region of interest) – esimerkiksi kasvojen, ihmiskehon tai auton

rekisterikilpien – peittäminen kuvanäkymästä ilman, että valvonnasta tulee tehotonta. [Dugelay *et al.* 2013] Käsittelen seuraavaksi erilaisia ehdotettuja yksityisyyssuodattimia käyttäen pohjana Winklerin ja Rinnerin [2010] luomia luokitteluja. Lisäksi teen katsauksen yksityisyyssuodattimien evaluointiin.

3.1. Tyhjäys

Tyhjäys (engl. blanking) on yksityisyyssuodatin, jolla huomioalueet leikataan kokonaan kuvasta [Winkler and Rinner 2010]. Tämä voi tapahtua siten, että koko huomioalue täytetään peitevärillä [Aved and Hua 2012] (kuva 1) tai ympäristöstä saatavalla datalla [Cheung and Zhao 2006]. Jälkimmäisessä tapauksessa videokuvaa katsova ei välttämättä edes huomaa, että valvonta-alueella on ketään, kuten voidaan huomata kuvasta 2: vasemmassa kuvassa näkyy mustaan pukeutunut henkilö, joka on kadotettu tyhjäyksellä oikeanpuoleisessa kuvassa.



Kuva 1 Tyhjätyn alueen täyttäminen värillä [Aved and Hua 2011]

Kuva 2 Tyhjätyn alueen täyttäminen taustalla [Cheung and Zhao 2009]

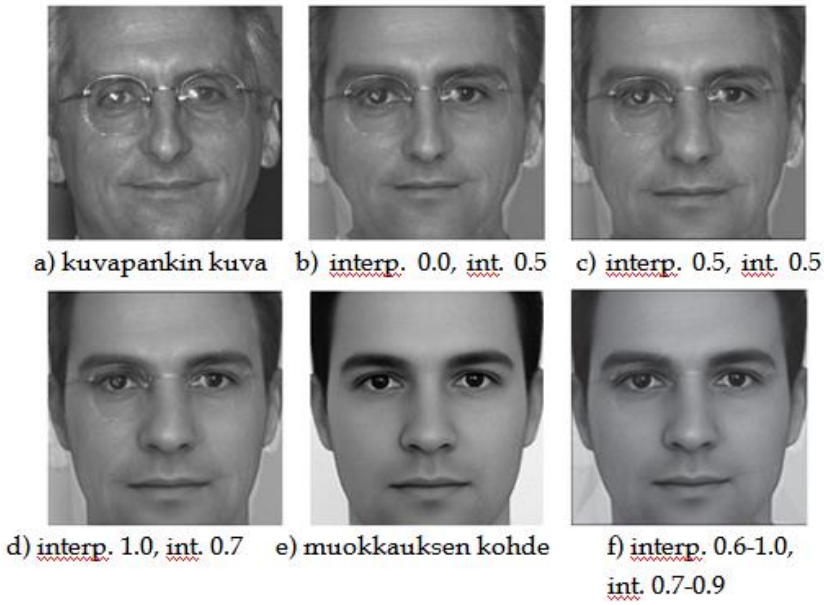
3.2. Hämäräperäistäminen

Tyhjäys poisti huomioalueet kokonaan näkyvistä kuvasta. Se on yksinkertainen tekniikka, mutta kuten kuvaesimerkkien perusteella voidaan havaita, se voi toisaalta hävittää valvonnalle tärkeää informaatiota: mukaan lukien esimerkiksi sen, mitä kuvattava tekee kuvausalueella.

a) b)

Kuva 3 Hämäräperäistäminen a) sumentamalla ja b) mosaiikiksi muuntamisella [Chinomi *et al.* 2008]

Kun tyhjentämisen sijasta käytetään *hämäräperäistämistä* (engl. obfuscating), voidaan säilyttää esimerkiksi informaatio kuvattavan käyttäytymisestä. Hämäräperäistäminen perustuu huomioalueen yksityiskohtien vähentämiseen. Perinteisimpiä hämäräperäistämistekniikoita ovat pikselöinti, mosaiikiksi muokkaaminen ja *sumentaminen* (engl. blurring). [Winkler and Rinner 2010.] Kuva 3 näyttää, miten sumentaminen ja mosaiikiksi muokkaaminen muuntavat kuvaa.



Kuva 4 Yhteensulautus eri voimakkuuksilla [Korshunov and Ebrahimi 2013]

Korshunov ja Ebrahimi [2013] esittävät puolestaan tekniikan, jossa kahden henkilön kasvokuvat *yhteensulautetaan* (engl. morphing). Järjestelmä hakee rajatusta kuvapankista kasvot, jotka on kuvattu sopivasta kuvakulmasta. Tämän kuvan ja monitoroitavan henkilön kasvojen yhteensulautus voidaan tehdä erilaisilla interpolointi- ja intensiteettivahvuuksilla. Interpoloinnissa kuvan pikselien järjestystä muutetaan: esimerkiksi niitä voidaan vähentää tai lisätä. Intensiteetti puolestaan viittaa kuvan värien kirkkaus- ja tummuusasteeseen. Yhteensulautuksen vaikutukset ovat nähtävissä kuvassa 4.

a) 10 % b) 50 % c) 100 %

Kuva 5 Sarjakuvaistaminen voimakkuuksilla [Erdélyi *et al*. 2014]

Erdélyi ja muut [2014] ehdottavat lisäksi kuva-alueiden *sarjakuvaistamista* (engl. cartooning). Sarjakuvaistamisessa sovelletaan sumentamista niin, että kohteiden rajat pyritään pitämään mahdollisimman selkeinä. Tekniikan vaikutukset videoon ovat nähtävissä kuvassa 5. Kuten on havaittavissa, kuva näyttää sitä sumeammalta, mitä voimakkaampaa sarjakuvaistamista käytetään. Tästä huoli-

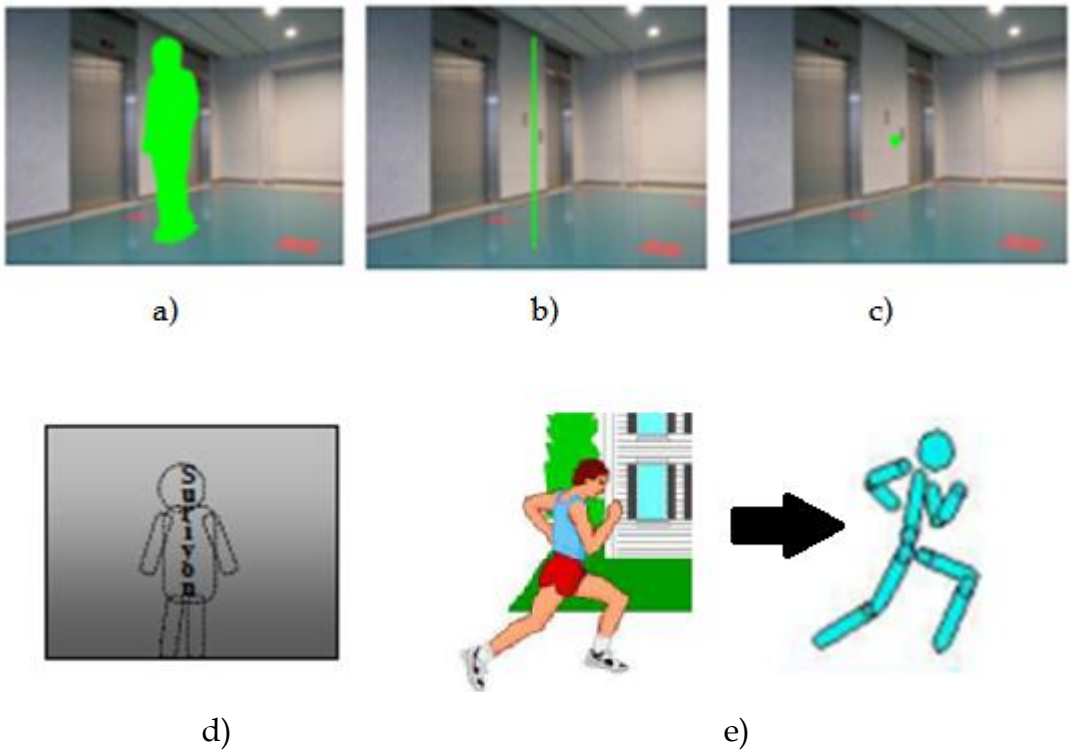
matta kuvan kohteiden suurpiirteiset rajat pysyvät selkeinä. Näin ollen sarjakuvaistamista voidaan käyttää koko kuvaan eikä pelkästään huomioalueisiin, mikä antaa tekniikalle etua muihin yksityisyyssuodattimiin verrattuna.

3.3. Abstraktointi

Aiemmissa yksityisyyssuodattimissa ajatuksena oli huomioalueiden poistaminen tai niiden tekeminen epäselkeiksi. Abstraktoinnissa puolestaan on tarkoituksena esittää kuvattava symbolisessa tai semanttisessa muodossa. Tämä toteutetaan siten, että huomioalueet korvataan kuvin tai metadatalla. [Winkler and Rinner 2010.]

Kuva 6 Abstraktointi käyttäen hymiöitä [Korshunov and Ebrahimi 2014]

Abstraktointia käyttävät yksityisyyssuodattimet voivat esimerkiksi peittää kuvattujen henkilöiden kasvot hymiöillä (kuva 6) [Korshunov and Ebrahimi 2014]. Heidät voidaan myös esittää siluetteina, pelkkinä pisteinä [Chinomi *et al*. 2008] tai tikku-ukkoina [Senior *et al*. 2003]. Metadataa hyödyntävät tekniikat voivat puolestaan näyttää valvottavat henkilöt esimerkiksi pelkkinä niminä [Tansuriyavong and Hanaki 2001] tai kaksi- ja yksiulotteisina mittoina ruudulla [Chinomi *et al.* 2008]. Näistä abstraktoinnin muodoista on nähtävissä esimerkkejä kuvassa 7.



Kuva 7 Abstraktointia. a) Siluetti, b) vertikaali viiva, c) piste [Chinomi *et al.* 2008], d) pelkkä kuvattavan nimi [Tansuriyavong and Hanaki 2001] ja e) tikkuukoksi muuttaminen [Senior *et al.* 2003]

3.4. Kryptaus

Edellä mainitut yksityisyyssuodattimet poistivat huomioalueet, mutta ne eivät antaneet keinoja muunneltujen kuvakohtien palauttamiseksi. Joissakin tapauksissa tämä ominaisuus ei häiritse kamerajärjestelmien alkuperäisen tavoitteen saavuttamista: esimerkiksi virtuaalivierailija voi tarkastella taidemuseon eri tiloja kameroiden välityksellä ilman, että hänen koskaan tarvitsisi nähdä paikallaolijoiden kasvonpiirteitä.

Kuten Ebrahimi ja Dufaux [2008] kuitenkin huomauttavat, on olemassa tilanteita, joissa kykenemättömyys alkuperäisten kuvien palauttamiseen voi huonontaa kamerajärjestelmien toimintaa. Esimerkiksi valvontakameroiden videomateriaalista ei pystyttäisi yksityissuodattimien takia saamaan selville, miltä rikoksentekijä näyttää. Tämä puute voi kirjoittajien mukaan vähentää operaattoreiden halukkuutta käyttää turvakamerajärjestelmiä, jotka muokkaavat videokuvaa yksityisyyssuodattimilla.

Ebrahimi ja Dufaux jatkavat ehdottamalla *sotkemista* (engl. scrambling), jossa huomioalueet kryptataan. Kryptauksessa he käyttävät hyödyksi kuvien pakkaukseen käytettäviä, yleisiä algoritmeja: esimerkiksi *diskreettiä kosinimuunnosta* (engl. discrete cosine transform).

Diskreetti kosinimuunnos poistaa kuvasta ne kohdat, jotka eivät tee kuvaan merkittäviä lisäyksiä ihmiskatsojan näkökulmasta. Algoritmi jakaa kuvan ensin 8x8 pikselin kokoisiin neliöihin alkaen vasemmalta oikealle ja ylhäältä alaspäin. Seuraavaksi jokaisen pikselin intensiteettiarvot muutetaan etumerkillisiksi: esimerkiksi jos kuvan pikseleiden alkuperäiset arvot ovat välillä 0 ja 255, jokaisesta arvosta vähennetään 128. Tämän jälkeen kuvatiedosto käännetään kuvaamaan avaruudellisuuden sijasta taajuuksia, jotka muodostetaan käyttäen painotettua kosinisummaa. Näin kuva jaetaan osiin niiden tärkeyden mukaisesti. Turhaksi oletettua dataa voidaan vähentää edelleen niin kutsutussa *kvantittamisvaiheessa* (engl. quantization). [Acharya and Tsai 2005.]

Sotkeminen käyttää hyödykseen sitä, että diskreetti kosinimuunnos kuvaa pikseliarvot etumerkillisinä lukuina. Pseudo-satunnainen numerogeneraattori luo siemenluvun, jonka perusteella diskreetin kosinimuunnoksen luomien arvojen etumerkit käännetään päinvastaisiksi sotkettavilla alueilla. Kryptattu huomioalue muistuttaa tulokseltaan hämäräperäistämistekniikalla tehtävää suodatusta (kuva 8). Se voidaan kuitenkin kääntää takaisin ihmisoperaattorin ymmärtämään muotoon, jos tiedetään siemenluku ja kryptatun alueen sijainti kuvassa.

Kuva 8 Ajoneuvo, joka on sotkettu [Ebrahimi and Dufaux 2008]

3.5. Yksityisyyssuodattimien tehokkuuden arviointi

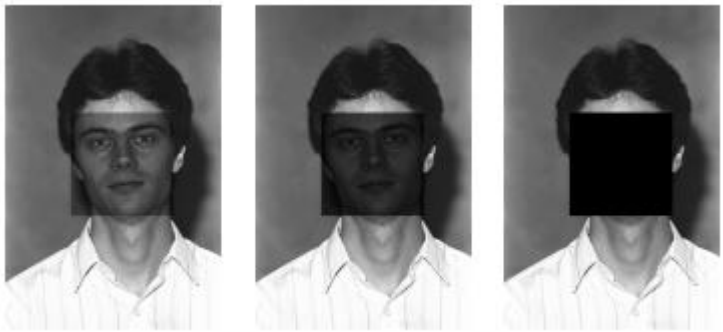
Korshunov ja muut [2013] huomauttavat, että yksityisyyssuodattimien kehittämisessä on olemassa selvä puute: niiden tehokkuuden arvioimismenetelmät ovat

vähäisiä. Toisaalta yksityisyyssuodattimien pitäisi kyetä piilottamaan kuvattavien henkilöllisyydet; toisaalta nämä toimenpiteet eivät saisi haitata kameravalvonnan toimintaa. Olisi tärkeää pystyä arvioimaan, mitkä yksityisyyssuodattimet pystyvät tarjoamaan parhaimman tasapainon valvonnan tehokkuuden ja yksityisyydensuojan välillä.

Korshunov ja muut jatkavat argumentoimalla, että koska yksityisyys on subjektiivinen käsite, myös yksityisyyssuodattimien arvioinnin kuuluisi perustua yksilöllisiin näkemyksiin. Siksi he lähestyvät yksityisyyssuodattimien evaluointia käyttäen joukkoistamista ja virtuaalista kyselyohjelmaa, VideoRatea. Valikoidut, vapaaehtoiset henkilöt kirjautuvat kyselyohjelmaan Facebook-tunnuksillaan, minkä jälkeen heille esitetään erilaisia videopätkiä. Videot käyttävät erilaisia yksityisyyssuodattimia, minkä lisäksi niissä olevat henkilöt voivat käyttäytyä joko normaalisti tai epäilyttävästi. Epäilyttäväksi käyttäytymiseksi lasketaan esimerkiksi silmien ja kasvojen peittäminen aurinkolaseilla ja/tai huivilla.

Jokaisen videopätkän jälkeen koehenkilöltä kysytään, mitä sukupuolta tai ihonväriä kuvattava edusti ja minkälaiset vaatteet hänellä oli päällään. Optimaalinen tulos olisi, että yksilön identifioivat piirteet olisivat tunnistamattomissa, kun taas hänen käyttäytymistään pystyttäisiin arvioimaan oikein; pahimmassa tapauksessa tulokset olisivat päinvastaiset. Korshunovin ja muiden tekemässä kyselyssä parhaiten pärjäsi pikselöinti, kun sitä verrattiin sumentamiseen ja siluetiksi muuntamiseen.

Dugelay ja muut [2013] toteavat, että vaikka subjektiiviset kyselyt voivat tuottaa hyvälaatuisia tuloksia, on niiden toteuttaminen käytännössä kallista ja aikaa vievää. Niinpä he ehdottavat objektiivista ja automaattista evaluointijärjestelyä: Kuvapankin kuviin asetetaan ensin yksityisyyssuoja automaattisesti, minkä jälkeen kasvojen paikantamis- ja tunnistusalgoritmit pyrkivät analysoimaan kuvia. Parhaassa tilanteessa kasvojen paikannus onnistuu, mutta henkilöä ei silti pystytä tunnistamaan. Testituloksissa arvioitiin pikselöintiä, sumentamista ja maskeerausta eri tehokkuustasoilla, minkä lisäksi kuvien henkilöt olivat eri etäisyyksillä kamerasta. Toisin kuin subjektiivisessa kyselyssä, Dugelayn ja muiden tulokset viittaisivat siihen, että paras tulos tulisi käyttäen 90-prosenttisesti peittävää maskeerausta. Maskeerauksessa huomioalueen päälle asetetaan läpikuultava, yksisävytteinen taso. Tästä yksityisyyssuodattimesta on nähtävissä eri vahvuisia esimerkkejä kuvassa 9.



Kuva 9 Maskeeraus vahvuuksilla a) 0.4, b) 0.7 ja c) 1.0 [Dugelay *et al.* 2013]

4. Yksityisyyden hallinta

Edellisessä luvussa esitellyt yksityisyyssuodattimet antoivat keinoja, joilla videodataa voidaan muokata yksityisyyttä tukeviksi. Yksityisyyssuodattimet ovat kuitenkin vain välineitä, jotka eivät määrittele, milloin niitä sovelletaan. Tässä luvussa käydään läpi mahdollisia vastauksia kysymykseen, miten valvottava henkilö voisi vaikuttaa itsestään tallennettavaan videomateriaaliin ja sen levittämiseen. Lisäksi pohdin näiden ratkaisumallien etuja sekä haasteita.

4.1. Yleisten yksityisyysasetusten kytkeminen päälle

Valvonta-alueet voivat tarjota kuvattaville mahdollisuuden kytkeä päälle yksityisyystilan, joka on samanlainen kaikille sitä toivoville.

Schiffin ja muiden [2009] ehdotuksessa yksityisyystila aktivoituu, jos henkilöllä on yllään visuaalinen tunnusmerkki: esimerkiksi keltainen hattu tai vihreät liivit. Tässä tapauksessa henkilön kasvot peitetään käyttäen ellipsinmuotoista tyhjäystä. Käytännössä muidenkin yksityisyyssuodattimien soveltaminen voisi olla mahdollista.

Tämän ratkaisun etuna näkisin sen yksinkertaisuuden valvottavan näkökulmasta; haittapuolena puolestaan voisi olla se, että kuka tahansa voisi käyttää tunnettuja visuaalisia tunnusmerkkejä hyväkseen. Jos kamerajärjestelmän yksityisyyssuodatin ei tue kryptausta, mahdollinen pahantekijä voisi käyttää kyseistä heikkoutta hyväkseen.

Brassil [2005] esittelee puolestaan Cloak-nimisen järjestelmän, jonka avulla valvottava voi kieltää itsestään otettujen videokuvien levittämisen eteenpäin. Tämä onnistuu kantamalla pientä yksityisyystilan aktivoivaa laitetta, PED:iä (engl. privacy enabling device). PED-laitteen toiminta muodostuu sisäisestä kellosta, ID-tunnisteesta, paikantimesta ja langattomasta datasiirtimestä. Kun henkilö kulkee valvotussa tilassa, PED-laite lähettää hänen kulloisenkin sijaintinsa varastoitavaksi tietokantaan.

Brassil kutsuu tätä erityistä tietokantaa *selvittämöksi* (engl. clearinghouse). Jos kameraoperaattori haluaa jakaa tietyn video-osuuden, hän pystyy tarkistamaan

selvittämöstä, antavatko kaikki kuvissa ilmestyvät henkilöt hänelle tähän luvan. Kuvamateriaali täytyy välittää edelleen *siistijälle* (engl. sanitizer) siinä tapauksessa, että yhdelläkin videolla olevalla henkilöllä on mukanaan aktivoitu PED-laite. Siistijä muokkaa videomateriaalin automaattisesti yksityisyystoiveita kunnioittavaksi. Tämän jälkeen kameraoperaattori voi jakaa video-osuuden eteenpäin.

Cloak-järjestelmän ajatus voisi olla hyvä varsinkin nykyaikaisessa maailmassa, jossa henkilöistä kerätyt tiedot voivat päätyä hallitsemattomasti kolmansille osapuolille. Järjestelmä pystytään lisäksi rakentamaan jo valmiiden kamerajärjestelmien ympärille. On kuitenkin kyseenalaistettavaa, kuinka valvottava voi varmistaa, että kameraoperaattori oikeasti toteuttaisi tekemänsä yksityisyyslupaukset.

Näiden ratkaisumallien ohella Wickramasuriya ja muut [2004] yhdistävät yksityisyysmallissaan *RFID-tunnisteita* (engl. radio frequency identification) ja liikkeentunnistimia. RFID-tunnisteet ovat pieniä laitteita, joita käytetään esimerkiksi tuotteiden etätunnistamiseen ja paikantamiseen. Tehtävä onnistuu radiotaajuisten kyselysignaalien avulla, jotka kulkevat RFID-tunnisteen ja sen lukijalaitteiden välillä. [Lindstrom and Thornton 2005.] Esimerkkejä RFID-tunnisteista on kuvassa 10.

Kuva 10 Erilaisia RFID-tunnisteita [Lindstrom and Thornton 2005]

Wickramasuriya ja muut [2004] käyttävät RFID-tunnisteita alueella työskentelevän henkilökunnan yksityisyyden suojelemiseen. Kaikille henkilökunnan jäsenille annetaan omat, henkilökohtaiset RFID-tunnisteensa, joita he pitävät mukanaan jatkuvasti työaikana. Tunnisteihin on lisätty esimerkiksi tietoa siitä, mitkä alueet ovat kullekin henkilökunnan jäsenelle sallittuja. Kun henkilökunnan jäsen

kulkee hänelle sallituissa tiloissa, kamerajärjestelmä muokkaa videokuvan yleisten yksityisyysasetusten mukaisesti. Jos henkilöllä ei puolestaan ole RFID-tunnistetta tai hän on liikkuu hänelle kielletyllä alueella, yksityisyyssuojausta ei käytetä kuvauksessa.

Esitetty ratkaisumalli voisi sopia varsinkin työskentelytiloihin, joissa henkilökunnalle on määritelty omat alueensa. Sen sijaan menettelytavassa ei nähdäkseni oteta huomioon tapauksia, joissa alueella liikkuisi paljon myös henkilökunnan ulkopuolisia. Esimerkiksi sairaalassa ja kaupassa vastaava lähestyminen ei välttämättä kävisi, jos potilaille ja asiakkaille haluttaisiin antaa myös mahdollisuus yksityisyyteen.

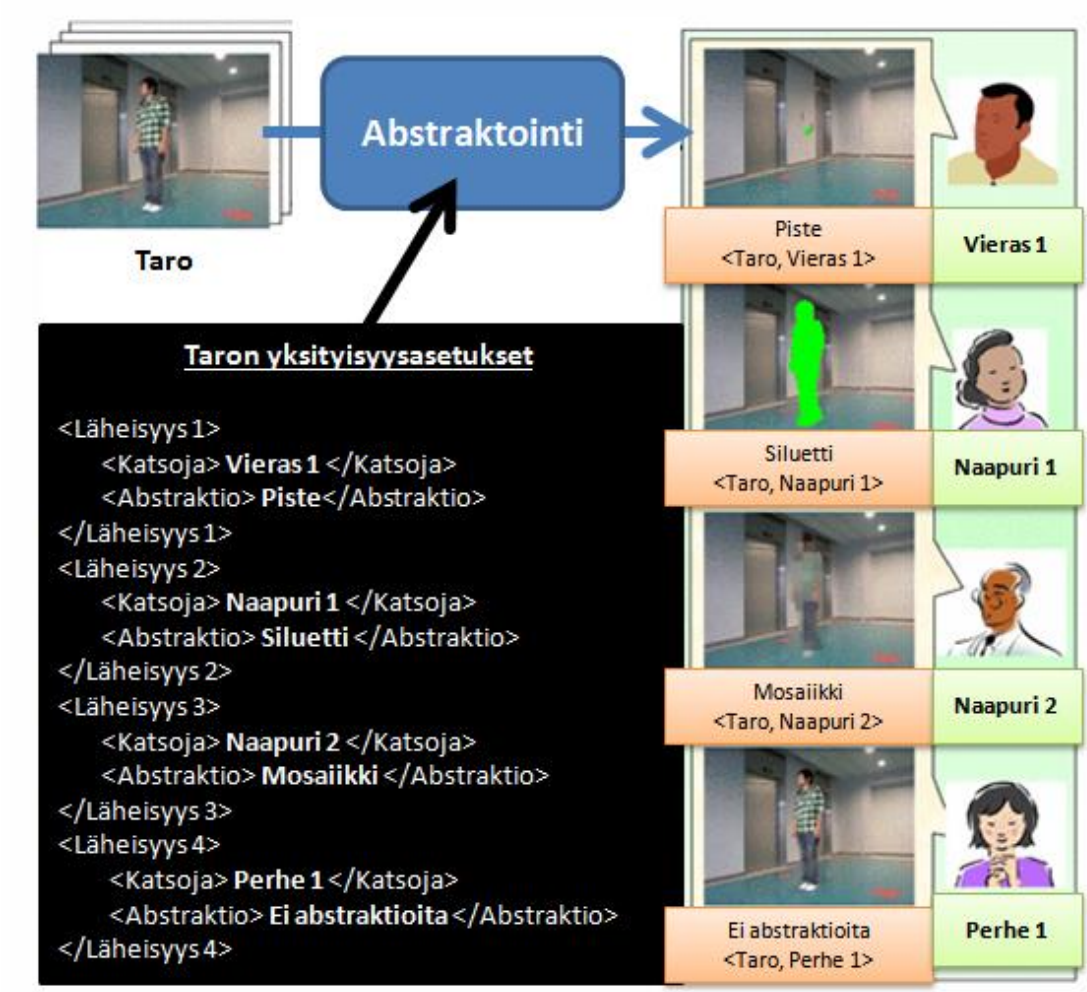
4.2. Henkilökohtaisten yksityisyysasetusten määrittäminen

Joissakin tapauksissa yleiset yksityisyysasetukset riittävät kaikkien osapuolten tarpeisiin. Kuitenkin on myös tilanteita, joissa on tärkeää, että valvottavat henkilöt voivat muodostaa uniikit, henkilökohtaiset yksityisyysasetukset itselleen.

Hongin ja Landayn [2004] Confab-järjestelmä antaa seurattaville henkilöille täyden hallinnan heistä kerättävästä datasta. Järjestelmässä ei käytetä keskitettyä tietokantaa, vaan jokaisesta henkilöstä tallennettava tieto säilytetään hänen omalla laitteellaan. Nämä tiedot voivat sisältää staattista informaatiota – kuten henkilön nimen – johon on liitettynä dynaamista dataa, esimerkiksi henkilön sijainti rakennuksessa ja se, mitä hän milläkin hetkellä on tekemässä.

Confab-järjestelmässä data on järjesteltynä *informaatiotiloihin* (engl. infospace), loogisiin tallennusyksikköihin, joihin voidaan muodostaa yhteys organisaation sisäisen verkoston kautta. Käyttäjä voi määritellä itse, minkä henkilökohtaisen datan hän haluaa näkyvän verkostossa julkisesti muille henkilöille ja laitteille. Salattu tieto näytetään puolestaan "tuntematon"-tekstinä hakukentässä. Hong ja Landay ehdottavat, että sama hakutulos näkyisi myös silloin, kun yhteyksissä on vikaa. Näin muille ei paljastuisi suoraan, että seurattava henkilö aktiivisesti kieltää kyseisen tiedon olevan julkista.

Hong ja Landay eivät tarkastele yksinomaan videotallenteiden käsittelyä järjestelmässään. Näkisin kuitenkin, että heidän ratkaisuaan voitaisiin mahdollisesti käyttää videoista suodatetun tiedon hallintaan. He ovat myös harvinaisia siksi, koska he ovat ottaneet huomioon kuvattavien toiveen pystyä kieltäytymään tietojensa jakamisesta passiivisella tasolla ilman, että heidän täytyisi kertoa muille, että he eivät halua tiettyjen tietojensa näkyvän muille. Jatkuva näennäinen yhteyksien vikatila voisi toki paljastaa tietojen salauksen; ratkaisu antaa kuitenkin mahdollisuuden piilottautua väliaikaisesti.



Kuva 11 PriSurv-järjestelmän toiminnan kuvaus [Chinomi *et al*. 2008]

Valvottavat pääsevät vaikuttamaan itsensä näkymiseen videokuvassa Chinomin ja muiden [2008] PriSurv-järjestelmää käyttäen. Seurattavat pystyvät määrittelemään yksityiskohtaisesti, miten eri henkilöt näkevät heidät omilta päätteiltään. Kuva 11 havainnollistaa järjestelmän toimintaa: Henkilö on määritellyt ensin omat yksityisyysasetuksensa, jotka sovellus kääntää XML-koodiksi. Kamerajärjestelmä pystyy tunnistamaan henkilön hänen kantamansa RFID-tunnisteen avulla, minkä ansiosta se kykenee käyttämään oikeita yksityisyysasetuksia. Seurattavan henkilön asetukset määrittelevät, että esimerkiksi tuntematon vieras näkee hänet pelkkänä pisteenä ruudulla; perheenjäsen puolestaan ilman minkäänlaisia yksityisyyssuodattimia.

PriSurv-järjestelmä ottaa hyvin huomioon sen, että yksilön käsitys omasta yksityisyydestään on monitasoinen. Järjestelmä antaa valvottavalle monipuolisesti valtaa päättää, miten hän ilmenee kuvissa eri katsojille. Toisaalta jäljelle jää kysymys, olisiko näin yksityiskohtaisten yksityisyysasetusten kirjoittaminen käyttäjälle mahdollisesti vaivalloista. Lisäksi järjestelmä ei pysty takaamaan,

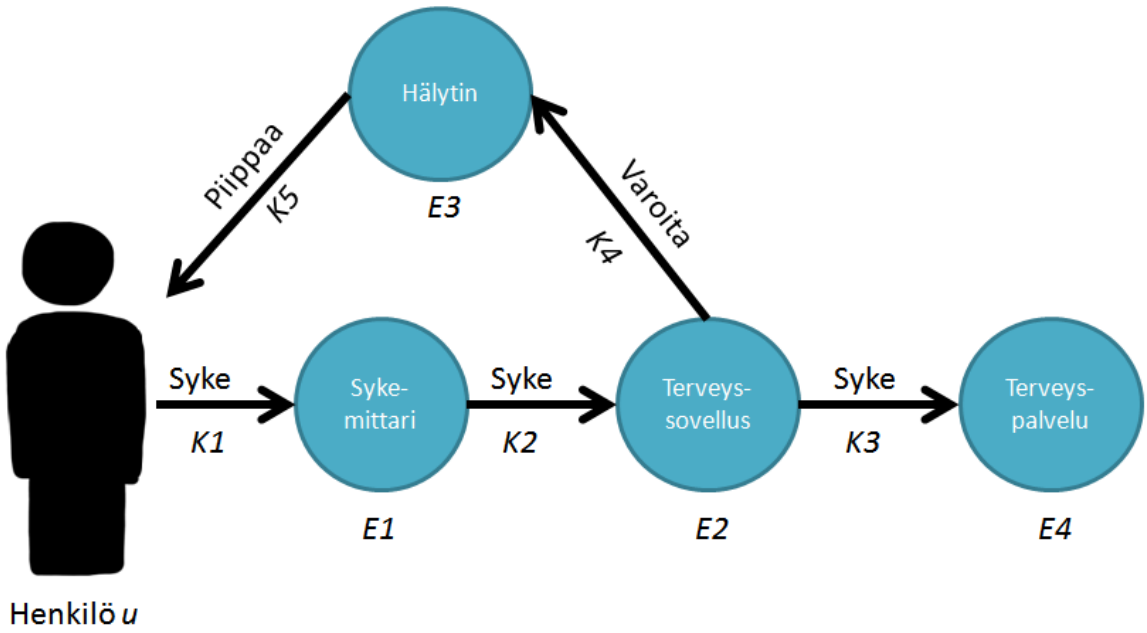
ettei joku käyttäisi toisten tunnuksia tai laitteistoja esiintyäkseen palvelussa eri katselijana.

5. Tiedonkulun seuranta

Edellä käsiteltiin tapoja, joilla hallita omaa yksityisyyttään. Ongelmaksi on vielä jäänyt, että kuvattavat eivät aina tiedä, ketkä kuvaavat heitä ja mistä syistä. Lisäksi he eivät tiedä, millä välineillä heitä kuvataan. Käsittelen tässä luvussa erilaisia ratkaisumalleja, joilla pyritään vastaamaan näihin kysymyksiin.

5.1. Valvonnan osapuolet, toimintatavat ja syyt

Könings ja Schaub [2013] miettivät tapoja mallintaa käyttäjälle, mitkä osapuolet seuraavat hänen toimintaansa, miten se tapahtuu ja miksi näin tehdään. Tämän lisäksi he pyrkivät kuvaamaan valvovia osapuolia. Valvovat osapuolet – laitteet, henkilöt ja organisaatiot – voivat esimerkiksi joko vain kerätä henkilöstä tietoja tai myös vaikuttaa suoraan henkilön elämään.



Kuva 12 Sykemittarin tiedon kulkeminen [Könings and Schaub 2013]

Kuva 12 esittää mahdollista yksityisyystietojen kulkua. Valvottavana on henkilö *u*. Häntä valvovat osapuolet on merkitty ympyröiden E1-E4 sisään. Nuolilla K1-K5 kuvataan tiedonkulkua ja niitä reaktioita, joita kerätty tieto voi laukaista ympäröivissä valvontalaitteissa: Henkilön *u* pulssilukemat siirtyvät ensin mittauslaitteen kautta terveydenhuoltoon tarkoitetulle sovellukselle, joka puolestaan

välittää tiedon eteenpäin terveyspalvelun tarjoajalle. Sovellus voi myös joissakin tapauksissa aktivoida hälyttimen, joka piipittää varoitukseksi terveydentilan huonontumisesta.

Näkisin, että esitelty ratkaisu voi havainnollistaa tiedon kulkemista. Könings ja Schaub tekevät lisäksi hyvän huomautuksen, että nykyaikaiset valvontavälineet eivät pelkästään seuraa henkilöitä passiivisesti: ne voivat myös vaikuttaa suoraan heidän elämiinsä. Köningsin ja Schaubin esittelemänsä esimerkit ovat kuitenkin varsin yksinkertaisia. Monimutkaiset, pitkät ja rönsyilevät tiedon kulkureitit voivat olla haastavia kuvata ymmärrettävällä tavalla.

5.2. Valvontavälineiden ohjelmistot

Winkler ja Rinner [2010] pohtivat, kuinka valvottavat voisivat kommunikoida kamerajärjestelmien kanssa ja näin saada niiden ohjelmistoista tietoja. Heidän järjestelmässään käyttäjä voi muodostaa yhteyden yksittäiseen valvontakameraan siten, että hän näyttää oman kännykkänsä tai tablettinsa ruudulta kaksiulotteisen viivakoodin kameralle. Kamera tulkitsee viivakoodin kyselyksi, jolla halutaan saada tietoja valvontalaitteen ohjelmistotilasta. Nämä tiedot haetaan kameran *PCR-lokista* (engl. platform configuration register), johon on kerätty tiedot kaikista kamerassa koskaan ajetuista ohjelmistokomponenteista. Käyttäjän ei ole tarkoitus lukea vaikeaselkoista PCR-lokia, vaan se lähetetään automaattisesti edelleen Internetissä olevalle palvelulle, TrustCenterille. TrustCenterillä on tietokanta, joka sisältää laitevalmistajien antamia tarkkoja tuotetietoja, esimerkiksi kameran ohjelmistojen lähdekoodin. PCR-lokin arvoja verrataan tietokannan sisältämiin tietoihin, joiden perusteella käyttäjälle luodaan käsitettävässä muodossa oleva raportti kamerasta. Raportissa esimerkiksi selostetaan kameran *käynnistyslataaja* (engl. bootloader) ja esitetään, kuinka kamera soveltaa yksityisyyssuodattimia kuvaan.

Winkler ja Rinner ottavat huomioon hyvin mahdolliset hyökkäysyritykset. Esimerkiksi *käkihyökkäykset* (engl. cuckoo attack) voisivat pyrkiä siihen, että luvattomasti aseteltu videokamera näyttäisi luotettavalta. Tämän Winkler ja Rinner ovat estäneet kyseisen hyökkäystavan eri varmennuksin. Toisaalta heidän suunnitelmansa onnistuminen perustuu kuitenkin pitkälti siihen, saadaanko valvontakameroiden valmistajat, kameraoperaattorit ja valvottavat osallistumaan järjestelmän kehitykseen.

6. Yleiset haasteet

Olen pohtinut edellisissä luvuissa lyhyesti eri yksityisyyden suojaamiseen tarkoitettujen menetelmien mahdollisia etuja ja haittoja. Tarkastelut kohdistuivat

kuitenkin lähinnä jokaisen ratkaisumallin omiin ominaisuuksiin ja toimintakuvioihin. Tarkoituksenani on näin ollen vielä tarkastella yksityisyyssuojausmenetelmien yleisiä haasteita.

Yksityisyyssuodattimet perustuvat huomioalueiden automaattiseen tunnistamiseen. Esimerkiksi kasvojen tunnistusalgoritmit kehittyvät jatkuvasti, mutta silti on kyseenalaistettavaa, toimivatko ne aina luotettavasti. Kuten Korshunov ja muut [2013] osoittavat, pienikin virhe yksityisyyssuodattimen asettelussa koituu suojauksen täydelliseen epäonnistumiseen. Toisaalta tunnistusalgoritmi voi tulkita ylimääräisiä kuva-alueita arkaluontoisiksi, jolloin kuvaa epäselkeytetään turhaan lisää.

Kysymyksenä on myös, mitkä alueet pitäisi lukea arkaluontoisiksi. Joissakin tapauksissa ihmisen kasvot nähdään tärkeimpinä tekijöinä tunnistamisessa; joissakin tapauksissa hänen koko kehonsa. Myös ihmisen ulkopuoliset tekijät, esimerkiksi auton rekisterikilvet, voivat paljastaa henkilöllisyyden. Näitä piirteitä pohdittiin edellä esitellyissä tutkimuksissa. Saini ja muut [2014] huomauttavat lisäksi, että kuvauksen aika, paikka ja konteksti voivat tietoina vaarantaa yksilön yksityisyyssuojan. Vaikka yksityisyyssuodattimia käytettäisiin, voi siis olla, että esimerkiksi pelkästään yhden sensuroimattoman henkilön tunnistaminen kuvassa voisi paljastaa ylimääräistä tietoa muista henkilöistä.

Epäilen kuitenkin, että kaikkien visuaalisten yksityisyysuhkien huomioonottaminen voi olla lähes mahdotonta. Jos kuvattavien identiteetin lisäksi poistettaisiin myös informaatio heidän sijainnistaan sekä käyttäytymisestään, päädyttäisiin käytännössä siihen tilaan, ettei kameroita olisi ollenkaan.

Tunnistusalgoritmit ja yksityisyyssuodattimet eivät aiheuta lisävaatimuksia pelkästään kameroiden ohjelmistoon vaan myös niiden fyysiseen puoleen. Hienostuneet ohjelmistot vaativat nimittäin kamerajärjestelmiltä muistia ja prosessointivoimaa. Kameroiden energiantarve myös kasvaa. Seuraukset eivät näin ollen sovi kamerajärjestelmien muihin päätavoitteisiin, edullisuuteen ja energiatehokkuuteen [Soro and Heinzelman 2009]. Koska yksityisyyssuojaan panostaminen voi nostaa kamerajärjestelmien valmistukseen ja käyttöön meneviä kuluja, valmistajat ja kameraoperaattorit voivat olla vastahakoisia kameroiden muuttamiseen.

Edellä mainituista syistä olisi tärkeää, että yksityisyyssuodattimien soveltamista käytännössä arvioitaisiin. Kuten Korshunov ja muut [2013] sekä Dugelay ja muut [2013] kuitenkin mainitsivat, etenkin suodattimien todellista yksityisyyssuojaa mittaavia menetelmiä on vähänlaisesti. Heidän ehdottamansa menettelytavat ovat lupaavia, mutta ne tulevat tarvitsemaan vielä jatkokehittelyä: evaluoinnit tuottivat hyvin erilaisia tuloksia myös silloin, kun vertailussa oli samoja yksityisyyssuodattimia.

Yksityisyyssuodattimien heikkoudet heijastuvat luonnollisesti myös kaikkiin niitä hyödyntäviin yksityisyyssuojausjärjestelmiin; tämänkaltaisilla järjestelmillä on kuitenkin myös muita haasteita. Useammassa tapauksessa saatettiin esimerkiksi hyödyntää RFID-pohjaista tekniikkaa henkilöiden tunnistamiseen ja paikantamiseen. Niin kuin Winkler ja Rinner [2010] kritisoivat, vastaavanlaiset ratkaisut eivät ota tarpeeksi hyvin huomioon niitä mahdollisia yksityisyys- ja tietoturvariskejä, jotka johtuvat RFID-laitteista.

Toisaalta katsoisin monen yksityisyysjärjestelmän ongelmaksi sen, että ne perustuvat usein optimistisuuteen. Kaikkien osapuolien odotetaan toimivan esitetyllä, toivotulla tavalla. Esimerkiksi valvontakameroiden valmistajilta ja kameraoperaattoreilta odotetaan avoimuutta ja rehellisyyttä. Heiltä toivotaan myös rahallista panostusta yksityisyyssuojien kehittämiseen sekä käyttöön. Heille ei kuitenkaan anneta vakuuttavia syitä toimia esitetyillä tavoilla. Valvottavilta puolestaan oletetaan aktiivisuutta kameravalvonnan seuraamiseen ja siihen perehtymiseen. Seuraamiseen tarvitaan joissakin tapauksissa välineitä, joita kaikilla valvottavilla ei välttämättä ole: esimerkiksi älypuhelimia. Joissakin muissa tilanteissa valvottaville tarjotaan välineistö, mutta tässäkin tapauksessa oletetaan, että valvottavat osaavat tai jaksavat oikeasti käyttää niitä.

Optimistisuuden lisäksi haasteena on esitettyjen järjestelmien ympäristöjen yksinkertaistaminen. Useassa ratkaisussa keskitytään yksittäisiin, yhteneväisiin kamerajärjestelmiin tai alueisiin, joissa eri kameraoperaattorit käyttävät samanlaisia yksityisyysmenetelmiä. Arvelen, että tämä ei vastaisi todellisuutta. Urbaanissa ympäristössä saattaa toimia monia toisistaan irrallisia kamerajärjestelmiä, joista kukin vahtii pienehköä, rajoitettua aluettaan. Yhteneväisen järjestelyn järjestelyksi vaadittaisiin mahdollisesti lakimuutoksia; muussa tapauksessa valvottava voisi turhautua nopeasti yksityisyyssuojausmenetelmien jatkuvaan vaihtumiseen.

Koska täydellinen yksityisyyssuoja tuntuu käytännössä mahdottomalta saavuttaa, suosittelisin pohtimaan, miten kameravalvonnan edut ja haitat saataisiin hyväksyttävään tasapainoon. Tämä vaatisi erilaisten turvallisuusriskien ja kamerajärjestelmistä saatavien etujen totuudenmukaista arvioimista.

7. Yhteenveto

Olen kuvaillut tässä katsauksessa yksityisyyttä ja kameravalvontaa. Vaikka nämä kaksi voivat olla ristiriidassa toistensa kanssa, tasapainon saavuttaminen ei ole välttämättä mahdottomuus.

Esittelemäni yksityisyyssuodattimet kuuluvat niihin keinoihin, joilla pyritään kunnioittamaan kuvattavien yksityisyyttä. Yksityisyyssuodattimien avulla voi-

daan poistaa parhaassa tapauksessa kuvattavien henkilöllisyydet siten, että heidän käyttäytymisensä tallentuu kameralle. Henkilötietojen palauttaminenkaan ei ole mahdottomuus tarvittaessa, kuten sotkeminen osoittaa.

Näiden lisäksi on kehitetty tapoja, joilla kuvattavat voivat määritellä omat yksityisyysasetuksensa. Joissakin tapauksissa yksityisyysasetukset ovat kaikille samanlaiset; joissakin tapauksissa valvottaville puolestaan sallitaan yksityisyysasetusten yksityiskohtainen muokkaaminen. Tähän käyttötarkoitukseen on käytetty erilaisia tekniikoita kuten PED-laitteita ja RFID-pohjaisia välineitä.

Tutkijat ovat pyrkineet myös selvittämään valvottaville, miten kamerajärjestelmät toimivat. Suunnitteilla on helposti ymmärrettäviä esityksiä tiedonkulusta sekä kameroiden fyysisistä ja ohjelmallisista ominaisuuksista. Tarkoituksena olisi tehdä valvonnasta avoimempaa ja käsitettävämpää.

Yksityisyyden suojelemisen tutkiminen on suhteellisen nuorta. Esitellyillä metodeilla on vielä monia ongelmia edessään. Yksityisyyssuodattimilta puuttuu esimerkiksi tehokkaat evaluointimenetelmät. Tekniset, ylimääräiset välineet voivat tuoda mukanaan uusia yksityisyyshaasteita. Menetelmissä ei ole myöskään otettu aina tarpeeksi hyvin huomioon suojauskeinojen tuomia rasitteita, jotka saattavat olla esteitä menetelmien todelliseen käyttöön.

Katsaukseni on suhteellisen suppea. Yksityiskohtaisen, perusteellisen metodien läpikäymisen sijaan pyrin luomaan yleissilmäyksen alaan. Jätin pois yksityisyyden kannalta keskeisiä teemoja kuten teknisen tietoturvan. Tämä voisi olla tulevaisuudessa myös kiinnostava ja tärkeä tutkimuksen aihe.

Pyrin tällä tutkielmalla ensisijaisesti lisäämään alan yleistietoutta sekä innostamaan ihmisiä osallistumaan valvontakamerajärjestelmien kehittämiseen. Valvontakameroiden yksityisyyskysymykset koskevat kaikkia – eivät pelkästään kameroiden valmistajia, kameraoperaattoreita, rikollisia ja poliiseja. Tästä syystä valvontakamerajärjestelmistä olisi hyvä olla avointa ja rakentavaa keskustelua.

Viiteluettelo

Acharya, Tinku and Tsai, Ping-Sing. 2005. *JPEG2000 Standard for Image Compression: Concepts, Algorithms and VLSI Architectures*. Wiley.

Aved, Alexander J. and Hua, Kien A. 2012. A general framework for managing and processing live video data with privacy protection. *Multimedia Systems* 18, 2, 123-143.

Bennett, Colin J., Haggerty, Kevin D., Lyon, David and Steeves, Valerie. 2014. *Transparent Lives: Surveillance in Canada*. Athabasca University Press.

Brassil, Jack. 2005. Using mobile *communications* to assert privacy from video surveillance. In: *Proc. of the 19th IEEE International Symposium on Parallel and Distributed Processing*. IEEE, 8-16.

Cheung, Sen-Ching S., Zhao, Jian and Venkatesh, M. Vijay. 2006. Efficient object-based video inpainting. In: *Proc. 2006 IEEE International Conference on Image Processing*. IEEE, 705-708.

Chinomi, Kenta, Nitta, Naoko, Ito, Yoshimichi and Babaguchi, Noboru. 2008. PriSurv: Privacy protected video surveillance system using adaptive visual abstraction. In: Shin'ichi Satoh, Frank Nack, Minoru Etoh (eds.), *Advances in Multimedia Modeling*. 144-154.

Dufaux, Frederic and Ebrahimi, Touradj. 2008. Scrambling for privacy protection in video surveillance systems. *IEEE Transactions on Circuits and Systems for Video Technology,* 18, 8 1168-1174.

Dugelay, Jean-Luc, Korshunov, Pavel, Melle, Andrea and Ebrahimi, Touradj. 2013. Framework for objective evaluation of privacy filters. In: *Proc. of SPIE Optical Engineering+ Applications*. International Society for Optics and Photonics.

Erdélyi, Adám, Barát, Tibor, Valet, Patrick, Winkler, Thomas and Rinner, Bernhard. 2014. Adaptive cartooning for privacy protection in camera networks. In: *Proc. of the 11th International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, 44-49.

Hong, Jason I. and Landay, James A. 2004. An architecture for privacy-sensitive ubiquitous computing. In: *Proc. of the 2nd International Conference on Mobile Systems, Applications, and Services*. ACM, 177-189.

Korshunov, Pavel, Cai, Shuting and Ebrahimi, Touradj. 2012. Crowdsourcing approach for evaluation of privacy filters in video surveillance. In: *Proc. of the ACM Multimedia 2012 Workshop on Crowdsourcing for Multimedia*. ACM, 35-40.

Korshunov, Pavel and Ebrahimi, Touradj. 2013. Using face morphing to protect privacy. In: *Proc. of the 10th International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, 208-213.

Kroener, Inga. 2014, *CCTV: A Technology Under the Radar?*. Ashgate Publishing.

Könings, Bastian, Schaub, Florian and Weber, Michael. 2013. Who, how, and why? Enhancing privacy awareness in ubiquitous computing. In: *Proc. of the International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops),* IEEE, 364-367.

Lindstrom, Pete and Thornton, Frank. 2005. *RFID Security*. Syngress Publishing.

Meyer, Sven and Rakotonirainy, Andry. 2003. A survey of research on context-aware homes. In: *Proc. of the Australasian Information Security Workshop Conference on ACSW Frontiers 2003-Volume 21*, ACSW Frontiers '03, 159-168.

Moncrieff, Simon, Venkatesh, Svetha and West, Geoff A. W. 2009. Dynamic privacy in public surveillance. *Computer* 42, 9, 22-28.

Saini, Mukesh, Atrey, Pradeep K., Mehrotra, Sharad and Kankanhalli, Mohan. 2014. W3-privacy: understanding what, when, and where inference channels in multi-camera surveillance video. *Multimedia Tools and Applications*, 68, 1, 135-158.

Schiff, Jeremy, Meingast, Marci, Mulligan, Deirdre K., Sastry, Shankar and Goldberg, Ken. 2009. Respectful cameras: Detecting visual markers in real-time to address privacy concerns. In: Andrew Senior (eds.), *Protecting Privacy in Video Surveillance*. Springer, 65-89.

Schneier, Bruce. 2013. *Carry on: Sound Advice from Schneier on Security*. Wiley.

Senior, Andrew, Pankanti, Sharath, Hampapur, Arun, Brown Lisa, Tian, Ying-Li and Ekin, Ahmet. 2003. *Blinkering surveillance: Enabling video privacy through computer vision*. Technical Report RC22886. IBM Research Division.

Soro, Stanislava and Heinzelman, Wendi. 2009. A survey of visual sensor networks. *Advances in Multimedia*, 2009, 1-22.

Tansuriyavong, Suriyon and Hanaki, Shin-ichi. 2001. Privacy protection by concealing persons in circumstantial video image. In: *Proc. of the 2001 Workshop on Perceptive User Interfaces*. ACM, 1-4.

Westin. Alan F. 1968. *Privacy and Freedom*. Bodley Head.

Westin, Alan F. 2003. Social and political dimensions of privacy. *Journal of Social Issues,* 59, 2, 431-453.

Wickramasuriya, Jehan, Datt, Mahesh, Mehrotra, Sharad and Venkatasubramanian, Nalini. 2004. Privacy protecting data collection in media spaces. In: *Proc. of the 12th Annual ACM International Conference on Multimedia*. ACM, 48-55.

Winkler, Thomas and Rinner, Bernhard. 2010. A systematic approach towards user-centric privacy and security for smart camera networks. In: *Proc. of the Fourth ACM/IEEE International Conference on Distributed Smart Cameras*, ACM, 133-141.

Winkler, Thomas and Rinner, Bernhard. 2014. Security and privacy protection in visual sensor networks: A survey. *ACM Computing Surveys (CSUR)* 47, 1, Article 2, 1-42.

Harmoninen haku metaheuristisena algoritmina

Andreas Valjakka

Tiivistelmä.
Harmoninen haku on yrityksen ja erehdyksen kautta toimiva algoritmi, jonka avulla raskaita laskentatehtäviä voidaan ratkaista kohtuullisessa ajassa ja riittävän hyvällä tarkkuudella. Yleisesti tällaisia metodeja kutsutaan heuristiikoiksi, joita korkeatasoisempia ovat monia heuristiikkoja yhdistävät metaheuristiikat. Pohjimmiltaan nämä laskentatehtävät ovat monesti optimointiongelmia, jotka koskevat jotakin käytännön elämän sovellusta. Tutkielmassani lähden liikkeelle optimoinnin perusteista ja etenen esittelemään harmonisen haun metaheuristisen toimintaperiaatteen sekä laskennallisen älykkyyden alan muita vastaavia algoritmeja.

Avainsanat ja -sanonnat: harmoninen haku, matemaattinen optimointi, heuristiikka, laskennallinen älykkyys, evoluutioalgoritmi

1. Johdanto

Optimointitehtävän lopputuloksena saadaan jotakin tiettyä tarkoitusperää vastaava paras mahdollinen tulos, joka voi olla mitä tahansa matemaattisen funktion minimiarvosta [Yang 2009] aina kustannustehokkaimpaan kaupungin vedenjakelujärjestelmään [Geem et al. 2001]. Perinteisesti optimointitehtävissä käytetään matemaattisia malleja, jotka takaavat parhaan mahdollisen tuloksen niille määritellyissä ideaalitehtävissä [Geem et al. 2001].

Harmoninen haku sekä muut luonnosta inspiraationsa saaneet mallit täydentävät heuristiikoilla matemaattisia malleja, joiden edellyttämä ideaali ei reaalimaailmassa usein toteudu. Tällaiset mallit monesti matkivat luonnossa esiintyviä prosesseja: harmoninen haku käyttää pohjanaan mallia jazzmuusikoiden improvisaatiosta. [Geem et al. 2001.]

Tutkielmani on kirjallisuuskatsaus harmonisesta hausta sekä sen käytöstä optimointilaskennassa. Aloitan perusteista ja jatkan teknisempiin aiheisiin, jotka liittyvät harmonisen haun rooliin optimointialgoritmien joukossa.

Toisessa luvussa esittelen lukijalle matemaattisen optimoinnin perusteita, joista annan yksinkertaisia ja perinteisiä esimerkkejä. Esittelen myös näiden menetelmien heikkouksia. Sen jälkeen perehdytän lukijan harmonisen haun toimintaperiaatteeseen.

Kolmannessa luvussa kerron tarkemmin heuristiikoista, jotka ovat aiheeni kannalta tärkeimpiä optimointimenetelmiä. Jatkan esittelemällä metaheuristii-

kat, jotka ovat korkean tason heuristiikkoja ja siten niiden luonnollinen jatke, sekä perustelen harmonisen haun metaheuristisuutta.

Neljännessä luvussa esittelen laskennallisen älykkyyden sekä sen piiriin kuuluvia tunnettuja luontoperustaisia algoritmeja. Keskityn esittelemään yhden niistä, evoluutiolaskennan, tarkemmin. Lopulta vertailen harmonisen haun toimintalogiikkaa evoluutioalgoritmien kanssa.

Viidennessä luvussa keskityn harmonisen haun saamaan kritiikkiin ja vastaukseen siihen. Lisäksi pohdin, kuinka perusteltua kritiikki on.

Kuudennessa luvussa kertaan lukijalle heuristiikkojen käyttöä optimoinnissa. Lopuksi tarkastelen harmonisen haun piirteitä muun muassa verrattuna muihin vastaaviin algoritmeihin.

2. Peruskäsitteitä

Tässä luvussa selitän matemaattista optimointia esittelemällä lukijalle kolme perinteistä optimointistrategiaa. Osoitan myös niiden puutteita. Sen jälkeen esittelen harmonisen haun toiminnan perusteet käyttäen apuna musiikista johdettua metaforaa ja näytän, kuinka ratkaisun tekeminen etenee hakua käyttäen.

Huomattakoon, että matematiikan kontekstissa sanan *optimointi* englanninkielinen käännös on joko *optimization* tai *programming*. Näistä jälkimmäinen ei viittaa tietokoneella tehtävään ohjelmointiin vaan on peräisin Yhdysvaltain armeijan *ohjelmasta* (program), jonka tarkoitus oli optimoida aikatauluja.

2.1. Matemaattinen optimointi

Matemaattinen optimointi on parhaan mahdollisen ratkaisun etsimistä matemaattisesti määriteltyyn ongelmaan [Snyman 2005], ja paras ratkaisu tarkoittaa etenkin käytännön sovelluksissa yleensä minimi- tai maksimiarvoa. Esimerkki sellaisesta on Hanoin kaupungin vedenjakelujärjestelmän kustannuksien minimoinen laskemalla, mikä on pienin tarvittava putkien lukumäärä [Geem et al. 2001]. Yhdenmukaisuuden vuoksi tutkielmassani kuvattavat esimerkit ovat aina minimointitehtäviä. Siitä seuraa, että näiden esimerkkien yhteydessä termit optimi ja minimi tarkoittavat samaa asiaa.

Optimointitehtävät ovat yleisesti muotoa

$$min\{ f(x): x \in X, g(x) \leq 0, h(x) = 0 \},$$

missä arvojoukko X on n-ulotteisen euklidisen avaruuden $\mathbb{R}^n$ osajoukko ja jonka arvot kuvautuvat reaaliluvuiksi. Relaatiot $x \in X, g(x) \leq 0$ sekä $h(x) = 0$ ovat optimointitehtävän *rajoitteita* (constraints), joiden puitteissa *kohdefunktio* (objective function) f ratkaistaan x:n suhteen. [Holder 1999.] Rajoitteita voi olla useampia, ja niiden yleinen esitys onkin

$$g_i(x) \leq 0, i = 1, 2, \dots, r$$

ja

$$h_j(x) = 0, j = 1, 2, \dots, s,$$

kun funktiot $g_i(x)$ tarkoittavat kohdefunktiota rajoittavia epäyhtälöitä ja funktiot $h_j(x)$ taas yhtäsuuruuksia [Snyman 2005].

Perinteisiä optimointitekniikoita ovat *lineaarinen* (linear), *nonlineaarinen* (nonlinear) sekä *dynaaminen optimointi* (dynamic programming) [Geem et al. 2001]. Lineaarisessa optimoinnissa ratkaistavan ongelman kaikki funktiot ovat lineaarisia. Nonlineaarisissa optimointiongelmissa vähintään yksi funktio ei ole sitä. Dynaaminen optimointi puolestaan perustuu optimaalisuusperiaatteeseen, jonka mukaan optimaalisin polku koostuu optimoiduista osapoluista. [Holder 1999.]

Lineaarinen optimointi voidaan kuvata lineaarisina suhteina. Toisin sanoen jos arvojoukko X on esimerkiksi kaksiulotteisen reaaliavaruuden osajoukko, voidaan nämä funktiot esittää tavallisessa kaksiulotteisessa (karteesisessa) koordinaatistossa suorina. Eräs algoritmi lineaaristen ongelmien ratkaisemiseen on Simplex, joka kokoaa kohdefunktion ja rajoitteet matriisiin, josta se laskee optimin hyödyntäen tietoa siitä, missä näiden funktioiden kuvaajat leikkaavat toisensa [Kyppö 2002].

Nonlineaariset optimointiongelmat reaalimaailmassa ovat yleisiä, ja niiden optimoiminen on paljon vaikeampaa kuin lineaaristen [Chinneck 2009]. Siitä seuraa, että nonlineaaristen ongelmien alue on laaja ja siksi ne ovat mielenkiintoisempia uusien optimointialgoritmien kannalta. Chinneck [2009] antaa esimerkiksi antennin signaalin heikkenemisen etäisyyden kasvaessa. Useimmiten nonlineaarisen optimoinnin kohdefunktiolle f voidaan löytää arvojoukosta useita *lokaaleja ratkaisuja* (local optimum), mutta *globaalin optimin* (global optimum) laskeminen on hankalaa [NEOS 2013]. Lokaali optimi tarkoittaa rajatulta alueelta löydettävää parasta ratkaisua, joita voi löytyä useampia riippuen alueen laajuudesta. Globaali optimi sen sijaan on yksikäsitteisesti koko optimointitehtävän paras ratkaisu. [Holder 1999.] Nonlineaarisia ongelmia voi alkaa ratkoa esimerkiksi redusoimalla ongelman lineaariseksi tai arvaamalla lähtöarvoksi jotakin mahdollisen optimin ympäriltä [Chinneck 2009] soveltamalla esimerkiksi derivaattaa. Nonlineaarisen ongelman mallintaminen lineaariseksi kuitenkin hävittää usein huomattavasti tietoa [Geem et al. 2001].

Dynaaminen optimointi on optimaalisten päätösten ketjun muodostamista rekursiivisesti. Yksittäinen päätös on *vaihe* (stage), jolle lasketaan jokin *tila* (state). Kun siirrytään seuraavaan vaiheeseen, käytetään aiemmassa vaiheessa laskettua tilaa uuden vaiheen tilan saamiseksi. Jokainen vaihe on riippumaton aiemmissa vaiheissa tehdyistä päätöksistä, sillä vain vaiheissa laskettuja arvoja kuljetetaan seuraaviin vaiheisiin. [Chinneck 2009.] Kyseistä ominaisuutta kutsutaan *optimaalisuusperiaatteeksi* (principle of optimality) [Holder 1999], joka toisin sanoen tarkoittaa tehtävän olevan mahdollinen ratkaista osissa. Menetelmää kutsutaan dynaamiseksi, sillä osassa sovelluksia vaiheet ratkaistaan suhteessa aikaan yleensä taaksepäin liikkuen, sillä alkutilanteesta on mahdollis-

ta luoda enemmän lopputuloksia kuin lopputilanteesta voi laskea alkutiloja. [Chinneck 2009.] On esimerkiksi mahdollista saada monta ratkaisua siihen, kuinka paljon viikon ajaksi tulisi ostaa elintarvikkeita kotitalouteen, mutta mahdollisuudet rajoittuvat huomattavasti, jos viikolle asettaa ensin budjetin.

2.2. Esimerkkejä optimointiongelmista

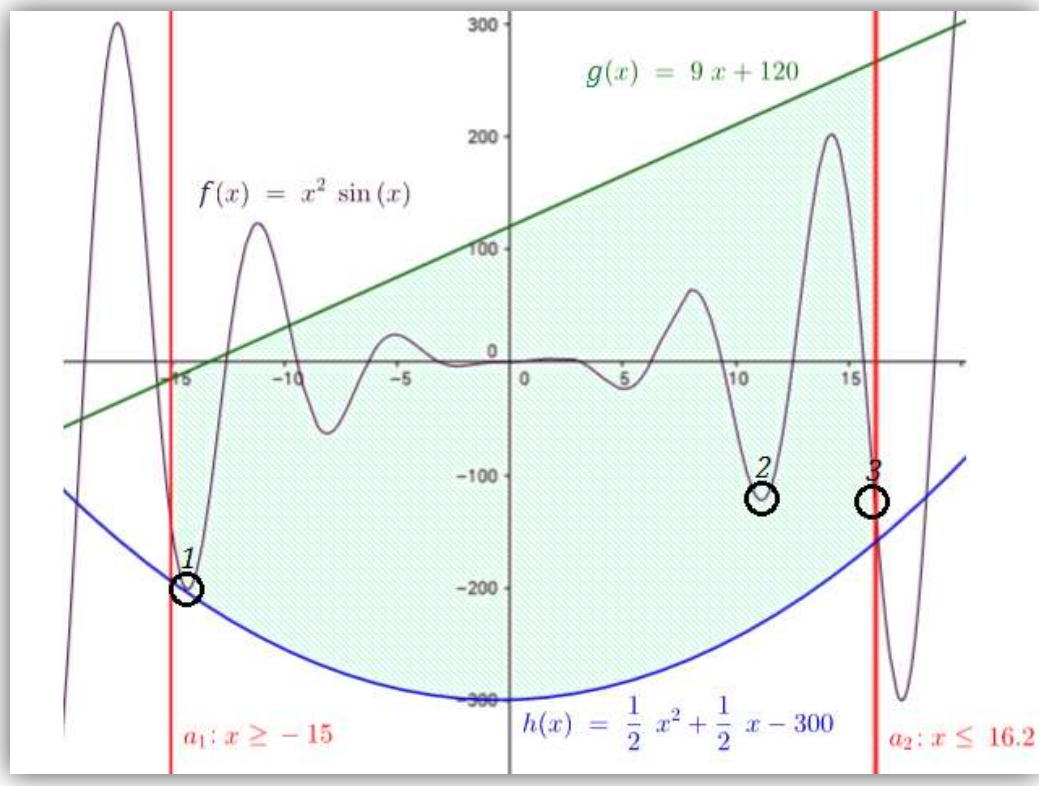
Lineaarisen optimointitehtävän ratkaisut (maksimi sekä minimi) voidaan löytää rajoitteita kuvaavien suorien leikkauspisteistä [Kyppö 2002], ja suorien leikkauspisteitä voidaan tutkia yksinkertaisesti muodostamalla niiden funktioista yhtälöryhmiä. Nonlineaariset ongelmat ovat kuitenkin reaalimaailmassa yleisempiä, joten aloitan suoraan niistä.

Kuvassa 1 esitetään kuvitteellinen nonlineaarinen minimointitehtävä, jonka kohdefunktiota

$$f(x) = x^2 \sin x$$

rajoittaa kaksi epäyhtälöä a_1 ja a_2, lineaarinen funktio $g(x)$ sekä nonlineaarinen funktio $h(x)$. Kuvaajan ratkaisut sijaitsevat siis väritetyn alueen sisällä, ja mahdollisia minimipisteitä ovat tässä tapauksessa 1, 2 ja 3. Kuva osoittaa, että on mahdollista saada kaksi lokaalia optimia (2 ja 3) koordinaatiston toisessa neljänneksessä, joista kumpikaan ei ole funktion globaali minimi (1). Tällaiseen ratkaisuun saattaa päätyä, jos muuttujan x arvoksi antaa vain positiivisia lukuja. Näistä lokaaleista optimeista ei myöskään voi nähdä suoraan, kumpi antaa pienemmän arvon. Ratkaisua laskeva algoritmi saattaa siis suorittaa turhaa laskentaa kahden sellaisen pisteen löytämiseksi, joista kumpikaan ei ole globaalisti oikea ratkaisu.

Eräs tapa ratkaista kuvan 1 mukainen tehtävä olisi esimerkiksi hyödyntää tietoa siitä, että derivaatan arvo valitussa pisteessä on sama kuin kulmakerroin. Kun kulmakertoimeksi saadaan nolla, on kyseessä tangentti, joka on x-akselin suuntainen ja joka täten on kohtisuorassa käyrän lakipisteeseen nähden. Toisin sanoen kyseisessä pisteessä kuvaaja on lokaalisti alimmillaan ennen kuin se kääntyy ja sen arvot alkavat jälleen kasvamaan. Tällaisia pisteitä on kuitenkin useita, ja kuvan 1 tapauksessa yksi minimeistä (3) ei noudata tätä sääntöä, sillä se ei ole kaaren lakipiste. Menetelmä myös edellyttää, että funktio on ääripisteissään derivoituva.



Kuva 1. Sinifunktion minimointitehtävä, jonka mahdolliset minimit ovat numeroituna.

Dynaamisen optimoinnin tehtävä muodostetaan vastaamalla ensin kysymyksiin kuten millaisia tiloja ratkaisussa esiintyy ja kuinka nämä tilat muodostuvat kussakin vaiheessa [Chinneck 2009]. Klassinen dynaamisen optimoinnin esimerkki on käyttää Dijkstran algoritmia [Dijkstra 1959] lyhyimmän mahdollisen reitin valitsemiseksi graafista [Chinneck 2009]. Graafin solmut on luontevaa jakaa osiin dynaamisesti optimoitaviksi siten, että jokaisesta solmusta edetään seuraavaan solmuun lyhintä matkaa pitkin. Solmut jaetaan käytyihin sekä vieraisiin, ja ne tallentavat tilakseen tiedon siitä, kuinka pitkä matka niihin on kuljettu. Kun algoritmia suoritetaan eteenpäin valikoimalla aina vieras solmu, johon löytyy lyhyin matka [Dijkstra 1959], tulee algoritmin suorituksen viimeiseksi solmuksi haluttu päätepiste. Algoritmi toimii optimaalisuusperiaatteen mukaisesti ja antaa oikean ratkaisun, mutta ongelman kasvaminen edellyttää rekursiivisten kutsujen lisäämistä, mikä kuormittaa muistia ja siten vähentää laskentatehoa. Vaikutus lisääntyy entisestään, kun ongelman koko kasvaa; Geem [2001] kutsuu ilmiötä *moniulotteisuuden kiroukseksi* (curse of dimensionality).

Kuvan 1 esittämä ongelma on nykyaikaisille tietokoneille ja jopa analyysin alkeet tuntevalle ihmiselle triviaali ja toimii vain esimerkkinä; optimointialgoritmien tehokkuutta mitataan käyttämällä erilaisia testifunktioita, joita ovat esimerkiksi Rosenbrockin sekä Michalewiczin funktiot [Yang 2009]. Niiden

avulla voidaan esimerkiksi vertailla eri algoritmeja tutkimalla, mikä saavuttaa parhaimman mahdollisen tuloksen minimaalisella määrällä iteraatioita.

2.3. Harmoninen haku

Harmoninen haku perustuu taulukkoon, jonka koon voi määritellä itse. Sen rivit edustavat optimointitehtävän ratkaisujoukkoja eli vektoreita, sarakkeet muuttujia sekä solut näiden muuttujien saamia arvoja. Rivien lukumäärää kuvataan *harmoniamuistin koolla* (harmony memory size, hms). Taulukko 1 havainnollistaa tätä musiikkimetaforan kautta: sarakkeiden muuttujat ovat muusikkoja, jotka tuottavat soluihin säveliä, ja näistä sävelistä syntyy riveille harmonioita. Taulukkoa kutsutaan harmoniamuistiksi (harmony memory, HM), ja se täytetään aluksi satunnaisilla arvoilla (sävelillä), jonka jälkeen uusia vektoreita (harmonioita) luodaan improvisoimalla eli muuttujat (muusikot) tuottavat uusia arvoja satunnaisesti. Mikäli tuotettu vektori antaa paremman ratkaisun kuin taulukon minimi (so. alin rivi), minimi poistetaan ja uusi vektori tallennetaan harmoniamuistiin. Muisti ylläpitää vektoreita paremmuusjärjestyksessä, ja improvisointia jatketaan, kunnes päätösehto (kuten iteraatiokertojen maksimin täyttyminen) saavutetaan. [Geem et al. 2001.]

	pianisti	**viulisti**	**kitaristi**	**tulos**
1. harmonia	A	C	E	paras[1]
2. harmonia	A	F	C	hyvä
3. harmonia	C	B	E	kehno
4. harmonia	B	B	D	huonoin

Taulukko 1. Harmoniamuistin rakenne.

Olkoon l taulukon leveys sekä hms sen korkeus eli muistissa säilytettävien ratkaisujen määrä. Yleisesti ilmaistuna harmoniamuisti muodostetaan antamalla sille kooksi $l * hms$. Taulukon 1 tapauksessa $l * hms = 4 * 4 = 12$. Täsmennettäköön, että esimerkiksi taulukon 1 tapauksessa käytettyjä nimekkeitä muuttujille ja riveille ei tallenneta tauluun, jota käytetään ohjelmallisesti. Käytännössä siis $l * hms$ ilmaisee yllä olevan taulukon tapauksessa arvoja sisältävien solujen määrän, ja näihin soluihin lasketaan myös tulokset.

Lisäksi harmoninen haku saa sovelluksesta riippuen [Ingram and Zhang 2009] muitakin parametreja, mutta Geem [2001] määrittelee niitä ainakin kaksi, jotka ovat *muistinkäyttöaste* (harmony memory consider rate, hmcr) sekä *sopeuttamisaste* (pitch adjusting rate, par). Näistä jälkimmäinen on valinnainen [Geem et. al 2001]. Parametrien tehtävänä on tuottaa muuttujissa hallittua satunnaisuutta uusien arvojen valintaan, mikä mahdollistaa globaalin optimin löytämi-

[1] Harmonia on paras, sillä se on puhdas kolmisointu (a-molli)

sen astumalla ulos niiden arvojen joukosta, jotka se on jo kertaalleen saanut. Kaikki Geemin [2001] esittämän alkuperäisen mallin parametrit on koottu yhteenvedoksi taulukkoon 3 luvun loppuun.

Ensiksi mainittu, $hmcr$, kuvaa sitä todennäköisyyttä välillä $[0, 1]$, jolla uuteen ratkaisujoukkoon valittava arvo etsitään niiden arvojen joukosta, jotka jo esiintyvät harmoniamuistissa [Geem et al. 2001]. Sen lisäksi se kuvaa, käytetäänkö ratkaisuista parhaita vai kaikkia [Yang 2009]. Arvon komplementti eli $\overline{hmcr} = 1 - hmcr$ on siis todennäköisyys sille, että muuttujan arvo on satunnainen [Weyland 2010]. Korkea $hmcr$-arvo saa algoritmin käyttämään harmoniamuistia mahdollisimman paljon ja hieman alhaisemmalla arvolla huonoimmat ratkaisut jäävät huomiotta. Arvojen vaihtelualue kuitenkin rajataan vain kelvollisiin alkioihin [Geem et al. 2001] riippuen siitä, miten optimointitehtävä määritellään. Esimerkiksi negatiivisia lukuja ei tulisi generoida, jos ratkaisuun etsitään vain positiivisia.

Toiseksi mainittu parametri on sopeuttamisaste par, joka muuttaa valikoitua muuttujan arvoa vielä kertaalleen johonkin lähistöllä olevaan arvoon [Geem et al. 2001; Yang 2009]. Yangin [2009] mukaan tyypillisimmin

$$hmcr \in [0{,}7;\ 0{,}95]$$

ja

$$par \in [0{,}1;\ 0{,}5].$$

Sopeuttamisastetta voi ajatella värähtelevänä kielenä, joka värisee sitä intensiivisemmin, mitä kauempana valikoitu alkio on optimia. *Värähtelyleveys* (bandwidth, b) kuvaa jatkuvaa aluetta, jonka arvoilla valikoitua alkiota voidaan sopeuttaa johonkin suuntaan ja kyseinen alue suppenee alkion lähestyessä optimiarvoa [Yang 2009]. Diskreeteille arvoille vastaava on *viereinen indeksi* (neighboring index, m) [Ingram and Zhang 2009].

Otetaan esimerkiksi yksinkertainen minimointitehtävä kohdefunktiolle, joka on muotoa

$$\min f(x) = (x_1 - 1)^2 + (x_2 - 2)^4 + (x_3 - 3)^6.$$

Sen optimaalisin ratkaisu on vektori $(x_1, x_2, x_3) = (1, 2, 3)$, jolloin $f(x) = 0$. Määritellään parametrin hms olevan kolme. Leveydeksi otetaan muuttujien ja tulossarakkeen yhteenlaskettu lukumäärä, joten harmoniamuistin koko on 12. Aluksi harmoninen haku luo satunnaisia ratkaisujoukkoja, esimerkiksi vektorit $(1, 1, 1)$, $(2, 2, 2)$ ja $(3, 3, 3)$. Taulukko 2 esittää harmoniamuistin lopputilannetta, jossa nämä alkuperäiset ratkaisujoukot eivät ole lihavoituna. Kuvaan seuraavaksi esimerkki-iteraatioita. Oletetaan, että ensin x_1:n arvoksi arvotaan harmoniamuistista 1. Toisessa vaiheessa x_1:n arvo muuttuu vielä kerran, jolloin siitä tulee 2. Sen jälkeen uusi muuttuja x_2 saa taulukosta arvoksi 3, eikä se muutu. Lopuksi muuttujan x_3 kohdalla arvo saadaan taulukon ulkopuolelta, ja siitä tulee 4, eikä se muutu. Näin generoitiin vektori $(2, 3, 4)$, joka asetetaan harmoniamuistiin. Algoritmi jatkaa iterointia luomalla lisäksi ratkaisujoukot $(1, 3, 3)$

ja lopputuloksen $(1, 2, 3)$. Taulukossa 2 esitetään kaikki esiintyneet ratkaisut niin, että kolme alinta ratkaisujoukkoa ovat muistista poistettuja.

	x_1	x_2	x_3	tulos
1. harmonia	**1**	**2**	**3**	**0**
2. harmonia	**1**	**3**	**3**	**1**
3. harmonia	2	2	2	2
poisto	**2**	**3**	**4**	**3**
poisto	3	3	3	5
poisto	1	1	1	65

Taulukko 2. Kaikki ratkaisujoukot. Lihavoidut saatiin algoritmin suorituksen tuloksena.

Yhteenveto eri parametreista ja käsitteistä on koottu taulukkoon 3. Toisin kuin puhtaasti matemaattinen optimointi, harmoninen haku ei perustu määrättyyn laskentatapaan. Harmonisen haun tarkoitus on laskea monia mahdollisia ratkaisuja, joihin se hyödyntää muistissa olevia (so. opittuja) tuloksia. Optimaalisen ratkaisun laskenta on siis kuin jazz-muusikoiden improvisointia, jossa paremmaksi oppii vain harjoittelemalla eli iteroimalla [Geem et al. 2001]. Optimoinnissa tällaisia metodeja kutsutaan heuristiikoiksi, joista kerron seuraavassa luvussa

lyhenne	**nimi**	**tehtävä**
HM	harmony memory, harmoniamuisti	tietorakenne (taulukko), jonne ratkaisut tallennetaan
hms	harmony memory size, harmoniamuistin koko	muistiin tallennettavien ratkaisujen (rivien) lukumäärä
hmcr	harmony memory consider rate, (harmonia)muistinkäyttöaste	todennäköisyys, jolla muuttuja saa taulukossa jo esiintyvän arvon
par	pitch adjusting rate, (sävelkorkeuden)sopeuttamisaste	todennäköisyys, jolla valittua muuttujan arvoa muunnetaan vielä kertaalleen
b/m	bandwidth/neighboring index, värähtelyleveys/viereinen indeksi	jatkuva väli/diskreetti arvo, jota par käyttää

Taulukko 3. Yhteenveto harmonisen haun parametreista.

3. Optimointia heuristiikoin

Heuristiikat (heuristics) ovat matematiikassa menetelmiä, joiden avulla pyritään optimaalisen ratkaisun löytämiseen ilman takeita siitä, että se onnistuu [Holder 1999]. Ne tarjoavat riittävän hyviä ratkaisuja kohtuullisella laskenta-ajalla sekä muistin kuormituksella [Geem et al. 2001; Chinneck 2009]. Esimerkiksi harmoninen haku mallintaa jazz-muusikoiden improvisointiprosessia [Geem et al. 2001]; sen alla olevan heuristiikan voisi kuvailla esimerkiksi niin, että kyseessä on mahdollisista alkioista etsittävä joukko, ja tähän joukkoon on suositeltavaa käyttää jo hyviksi todettuja alkioita, mutta liikkumisen varaa on. Kyseinen liikkumavara on myös heuristiikka, sillä se mallintaa värähtelevää kieltä. Näin siis harmoninen haku perustuu vähintään kahteen heuristiikkaan. Olennaista on huomata, että nämä heuristiikat ovat läheisesti sidottuja mallintamaansa ilmiöön.

Heuristiset mallit siis vaihtavat ehdottoman tarkkuuden ja tarkat alkuarvot kelvolliseen ratkaisuun, ja Yangin [2009] sanoin heuristiikat toimivat yrityksen ja erehdyksen kautta aivan kuten harmoninen haku laskee optimiratkaisun iteroimalla erilaisia harmonioita. Heuristiikkojen etu luvussa 2 esitettyihin matemaattisiin malleihin verrattuna on esimerkiksi se, etteivät ne edellytä ongelman redusoimista lineaariseksi. Sen lisäksi heuristiikat eivät edellytä tarkkaa alkuarvojen valitsemista. [Geem et. al 2001.]

Holder [1999] erottaa heuristisessa haussa matemaattisen ja tekoälyn näkökulman: matemaattisessa optimoinnissa heuristiikat ovat mitä tahansa menetelmiä optimin laskemiseksi, kun taas tekoälylle heuristiikat ovat algoritmeja, jotka soveltavat algoritmin suorituksen aikana arvoja laskevia heuristisia funktioita. Matematiikassa siis heuristiikat eivät itsessään ole algoritmeja vaan jopa niiden vastakohtia [Holder 1999]. Ne kuvaavat malleja, joiden perusteella algoritmeja voidaan suunnitella. Arkikielessä ajatusta vastaa heuristiikalle synonyyminen ilmaisu *nyrkkisääntö* (rule of thumb): algoritmin lähtökohdaksi voidaan ottaa yleispätevä luonnehdinta ottamatta kantaa varsinaisen algoritmin toteutukseen. Tekoälyn tapauksessa nyrkkisääntöjä edustavat heuristiset funktiot [Holder 1999], sillä ne laskevat nykyisen tilan, jonka perusteella algoritmi etenee. Esimerkiksi *ahneet* (greedy) algoritmit perustuvat nyrkkisääntöön, jonka perusteella optimiarvo saadaan *lyhytnäköisellä* (myopic) algoritmilla valikoimalla alkioita, jotka aina kasvattavat tulosta eniten kohti haluttua [Holder 1999]. Muun muassa toisessa luvussa esitelty Dijkstran algoritmi on tällainen.

3.1. Metaheuristiikat

Heuristiikat eivät välttämättä saavuta optimointitehtävän parasta globaalia ratkaisua operoidessaan omalla määrittelyjoukollaan vaan antavat tulokseksi lokaalin optimin, jonka voi laskea annetuista lähtöarvoista [Yang 2009]. Esimer-

kiksi harmonisen haun värähtelevän kielen kohdalla on helppo huomata, että heuristiikka on hyvin spesifi. Heuristiikat ovat siis Yangin [2010] sanoin deterministisiä, toisin kuin *metaheuristiikat* (metaheuristics). Ne ovat Yangin [2009] määritelmän mukaan useita heuristiikkoja yhdistäviä korkean tason algoritmeja, joiden tarkoitus on kiertää deterministisyys hallitulla satunnaisuudella. Niitä voi kuvailla myös hakuprosessia ohjaavina strategioina [Blum and Roli 2003]. Siten alkuarvot voivat muuttua algoritmin suorituksen aikana, minkä vuoksi metaheuristiset algoritmit soveltuvat globaalien optimien laskemiseen sisällyttämällä itselleen luonnolliselle maailmalle ominaista sattumanvaraisuutta [Yang 2010]. Olennainen ero tavanomaisiin heuristiikkoihin on niiden ongelmariippumattomuus, ts. ne eivät ole niin spesifejä kuin tavanomaiset heuristiikat.

Yangilla [2009] heuristiikkojen ja metaheuristiikkojen ero on kohtuullisen selkeä, mutta hän itse myöntää [2010], että termejä käytetään usein synonyymisinä. Esimerkiksi Holder [1999] ei puhu lainkaan metaheuristiikoista vaan heuristisen haun strategioista, jotka usein perustuvat luontometaforaan. Tutkielmassani nojaan Yangin [2009] luomaan jakoon, joka erottaa heuristiikat ja metaheuristiikat toisistaan. Vaikka Yang [2010] samastaa metaheuristiikat luontoperustaisten algoritmien kanssa, voidaan luontoperustaiset metaheuristiikat nähdä vain yhtenä ryhmänä. Esimerkiksi Blumin ja Rolin [2003] mukaisesti jako alkuperän perusteella luonnosta ja ei-luonnosta kumpuaviin on vain yksi tapa, jota he eivät pidä tarkoituksenmukaisena muun muassa sen tulkinnanvaraisuuden takia.

Yang [2009] esittää kolme metaheuristiikoille ominaista piirrettä, joista kaksi ovat algoritmeihin sisällytettyjä komponentteja. Ensiksi on *vaihtelevuus* (diversification), joka on algoritmisesti tuotettua luonnolle ominaista satunnaisuutta. Toinen piirre on *tehostuminen* (intensification), jonka mukaisesti algoritmi etsii ratkaisua tehokkaammin oikean ratkaisun ympäriltä. Nämä kaksi ovat metaheuristiikoille tärkeitä piirteitä, jotka mahdollistavat sekä hakuavaruuden laajan hyödyntämisen että mahdollisimman nopean konvergenssin eli lopputulokseen pääsemisen [Yang 2009]. Yang [2009] kutsuu näitä myös eksploraatioksi ja eksploitaatioksi, tässä järjestyksessä. Kolmas piirre kuvailee yleisluontoisemmin algoritmin toimintamekanismia, joka perustuu joko *liikeratään* (trajectory-based) tai *populaatioon* (population-based) [Yang 2009]. Karkeasti niiden ero on, että liikerataan perustuvat algoritmit käsittelevät yhtä ratkaisujoukkoa kerrallaan ja populaatioon perustuvat useampia.

Esitin jo aiemmin, että harmoninen haku käyttää vähintään kahta heuristiikkaa. Sen lisäksi se soveltaa uusiin ratkaisujoukkoihin muistinsa osaratkaisuja, joita hyödyntäen uusi vektori luodaan [Geem et al. 2001]. Harmoniamuistin

eksploitaation seurauksena ratkaisut siis tehostuvat; tähän vaikuttaa parametrin *hmcr* arvo eli se, missä määrin harmoniamuisti otetaan uusissa ratkaisuissa huomioon. Muistinkäyttöaste vaikuttaa kuitenkin ennen kaikkea lokaalilla alueella [Yang 2009]; vaihtelevuutta syntyy, kun satunnaisuus tuotetaan jokaiselle vektorin alkiolle kahdella tasolla. Siitä vastaavat jonkin verran sopeuttamisaste *par* ja enemmän toteutustavasta riippuva valinnainen satunnaistaja, joka voi yksinkertaisimmillaan vain arpoa uuden arvon [Yang 2009]. Sen lisäksi muistista valitun arvon satunnainen muutos suoritetaan sitä pienempänä, mitä parempi ratkaisu on jo olemassa [Yang 2009], toisin sanoen esimerkiksi värähtelyleveyden b arvoväli kutistuu eli eksploraatio vähenee sitä mukaa, kun lopputulos on lähellä optimia. Näin ollen harmoninen haku myös sisältää nämä metaheuristiikoille kaksi tärkeätä komponenttia. Lopuksi se on selkeästi populaatioon perustuva, sillä uusia vektoreita luodaan aina hyödyntäen harmoniamuistia joko kokonaan tai osittain.

3.2. Tunnettuja metaheuristiikkoja

Metaheuristiikat ovat eräs tehokkaimmista keinoista laskea kelvollisia optimeja. Sen vuoksi useita sellaisia on kehitelty viimeisten muutaman vuosikymmenen aikana. [Manjarres et al. 2013.] Ne perustuvat aina Yangin [2009] esittämiin kolmeen piirteeseen. Ero eri metaheuristiikkojen välillä on tehostamisen ja vaihtelevuuden tasapainotuksessa, hakutavassa tai taustailmiössä. Koska metaheuristiikat perustuvat erilaisiin ilmiöihin, ne sopivat paremmin tai huonommin johonkin tiettyyn sovellukseen. [Manjarres et al. 2013.] Ei siis ole olemassa kaikkiin tilanteisiin parhaiten sopivaa algoritmia. Manjarresin [2013] mukaan kuitenkin harmoninen haku on osoittanut käytännössä olevansa muita metaheuristiikkoja potentiaalisempi ja tehokkaampi useampaan tapaukseen.

Harmonisen haun suosio on ollut huomattavaa. Erääksi syyksi arvellaan sen suhteellista yksinkertaisuutta, joka on mahdollistanut esimerkiksi johdannaisten kehittämisen. [Manjarres et al. 2013.] Nämä johdannaiset ovat hybridejä, joita saadaan esimerkiksi yhdistämällä harmoniseen hakuun muita metaheuristiikkoja, muun muassa simuloitua jäähdytystä [Askarzadeh 2013]. Lisäksi Geem [2001] mainitsee muun muassa tabuetsinnän ja evoluutioalgoritmit. Erilaisista yksittäisistä metaheuristiikoista kiinnostuneille löytyy kattava, joskaan ei tyhjentävä, lista Yangin [2010] teoksesta.

Simuloitu jäähdyttäminen (simulated annealing, SA) on eräs vanhimmista ja suosituimmista metaheuristiikoista [Yang 2010]. Se perustuu yksittäiseen liikerataan, jonka toiminnan esikuvana on ollut metallin päästäminen eli kontrolloitu kuumentaminen ja viilentäminen, mikä parantaa metallin ominaisuuksia. Aluksi lämpötilaparametrille T annetaan mahdollisimman suuri arvo, jotta algoritmi tutkisi optimointiongelmaa mahdollisimman laajalti. Lopulta lämpötila

laskee, kun optimointitehtävästä löytyy esimerkiksi suuri pudotus minimiin. [Askarzadeh 2013.] Simuloitua jäähdyttämistä voi ajatella epätasaiselle pinnalle pudotettuna pallona, joka pomppii ympäriinsä epätasaisuuksien takia ja laskeutuu lopulta suurimpaan kuoppaan [Yang 2010].

Tabuetsintä (tabu [sic] search) on ensimmäinen muistia hyödyntänyt algoritmi, joka on tehokas ja usein käytetty [Yang 2010]. Algoritmi suorittaa liikesarjoja eri ratkaisujen hakemiseksi. Jotkin siirrot tosin luokitellaan kielletyiksi eli tabuiksi, ja ne ovat algoritmin tapa välttyä lokaalilta optimilta. [Geem et al. 2001.] Myös tabuetsintä perustuu liikerataan; se ei käytä muistia osaratkaisuiden tallentamiseen kuten harmoninen haku vaan kiellettyjen siirtojen välttämiseen.

Nämä kaksi yhdessä harmonisen haun kanssa antavat hyvän kuvan metaheuristiikkojen luonteesta. Harmoninen haku on yhdistetty myös muun muassa parviälykkyyden kanssa [Pandi and Panigrahi 2011]. Geem [2001] kirjoittaa lisäksi evoluutioalgoritmeista. Näitä kahta käsittelen tässä tutkielmassa laskennallisen älykkyyden paradigmoina, jotka voidaan nähdä joukkona erilaisia metaheuristiikoita. Syvennyn niihin lähemmin seuraavassa luvussa.

4. Laskennallinen älykkyys

Laskennallinen älykkyys (computational intelligence, CI) on *tekoälyn* (artificial intelligence, AI) osa-alue. Se tutkii mekanismeja, jotka mahdollistavat älykkään toimimisen muuttuvassa ympäristössä, ja parhaimpia tuloksia on saatu kehittämällä luontoa mallintavia algoritmeja. [Engelbrecht 2007.] Kuten luvussa 3 sivuttiin, tekoälyn näkökulmasta heuristiikat soveltavat algoritmeja, jolloin ne ovat määrättyjä ja siten tietyssä mielessä jäykkiä (vrt. matematiikkaan, jonka näkökulmasta heuristiikat ovat yleisluontoisia ohjenuoria). Toteuttaakseen heuristisuuden tuomia etuja matemaattisiin malleihin nähden CI-algoritmien tulee mallintaa älykästä toimintaa esimerkiksi oppimalla ympäristöstään ja sopeutumalla siihen [Engelbrecht 2007].

Käsitteinä metaheuristiikka ja CI voivat aiheuttaa sekaannusta: muun muassa Engelbrecht [2007] ei puhu lainkaan metaheuristiikoista, eivätkä esimerkiksi Yang [2009; 2010] tai Manjarres [2013] mainitse milloinkaan laskennallista älykkyyttä, vaikka he kaikki kirjoittavatkin samoista asioista. Voidaan esimerkiksi nähdä, että laskennallinen älykkyys on metaheuristiikkoja koskeva ala, sillä määritelmänsä mukaisesti CI tutkii älykkääseen toimintaan johtavia mekanismeja [Engelbrecht 2007]. Laskennallisen älykkyyden alalla ei kuitenkaan oteta kantaa ratkaistavien ongelmien tyyppiin: esimerkiksi Googlen DeepDream on ohjelma, joka visualisoi algoritmin suorittamaa kuvaluokittelua. Algoritmi on kuvia kerroksittain tutkiva *keinotekoinen neuroverkko* (artificial neural

network, ANN). [Mordvintsev et al. 2015.] Heuristiikat, siis myös metaheuristiikat, ovat puolestaan puhtaasti optimointilaskentaan käytettäviä menetelmiä.

Metaheuristiikat sekä laskennallinen älykkyys ovat ilmeisesti molemmat yhtä päteviä kattotermejä joukolle erilaisia luonnossa esiintyviä prosesseja mallintavia laskentamenetelmiä. Luulisin kyseen olevan pitkälti käyttökontekstista, ja koska tutkielmani koskee optimointia harmonisen haun avulla, voidaan kaikkia tässä luvussa käsiteltyjä algoritmeja ajatella metaheuristiikkoina. Kolmas hyvin samankaltainen termi on *pehmeä laskenta* (soft computing), jolla tarkoitetaan epäeksaktiutta sietäviä laskentatapoja [Zadeh 1994]. Päällekkäisyyksien välttämiseksi en keskity asiaan pehmeän laskennan kannalta.

4.1. Laskennallisen älykkyyden paradigmoja

Engelbrecht [2007] nimeää viisi laskennallisen älykkyyden pääasiallista paradigmaa, jotka ovat

- *neuroverkot* (neural networks, NN),
- *parviälykkyys* (swarm intelligence, SI),
- *keinotekoiset immuunijärjestelmät* (artificial immune systems, AIS),
- *sumeat järjestelmät* (fuzzy systems, FS) ja
- *evoluutiolaskenta* (evolutionary computation, EC).

Manjarres [2013] luokittelee metaheuristiikkoja samankaltaisiin kategorioihin perustuen niiden menettelytapaan; hän tunnistaa esimerkiksi evoluutiolaskennan sekä immuunijärjestelmät. Myös Yang [2010] lähestyy asiaa näin ja mainitsee esimerkiksi parvioptimoinnin, neuroverkot ja geneettiset algoritmit.

Neuroverkoilla luodaan järjestelmiä, jotka esimerkiksi kykenevät yhtäaikaisesti rinnakkaiseen laskentaan ja hahmontunnistukseen kuten ihmisaivot. Neuroverkon perusyksikkö on *keinotekoinen aivosolu* (artificial neuron), joka biologisen neuronin tavoin kokoaa eteenpäin välitettävän signaalin ympäristönsä ärsykkeistä, joita voivat olla esimerkiksi muiden aivosolujen lähettämät signaalit. [Engelbrecht 2007.] Neuroverkot voidaan ajatella metaheuristiikoiksi, sillä ne pyrkivät minimoimaan virheitä omassa oppimisessaan [Yang 2010], niin sanotusti optimoimaan oppimistaan. Neuroverkkoja voidaan opettaa (supervised learning) ja ne voivat lopulta oppia itse (unsupervised learning) erilaisten funktioiden avulla. Esimerkiksi Hebbin oppimissäännön mukaan aivosolu oppii lähettämään signaaleja mikäli muut aivosolut jatkavat saman signaalin viemistä eteenpäin. [Engelbrecht 2007.]

Parviälykkyyden perustana on sosiaalisten eläinten käyttäytyminen, esimerkiksi muurahaisparvien tapa jättää hajujälki ravinnon luo johtaviin polkuihin [Engelbrecht 2007]. Tunnettuja parviälykkyyttä edustavia metaheuristiikkoja ovat yllä kuvailtu *muurahaisyhdyskuntaoptimointi* (ant colony optimization, ACO) ja *hiukkasparvioptimointi* (particle swarm optimization, PSO) [Manjarres et al. 2013]. Hiukkasparvioptimoinnin taustalla on lintuparvien liikkuminen

[Yang 2009]: vaikka yksittäiset linnut (particles) lentävät eri radoilla ja välillä hieman eri suuntiin, kokonaisuutena parvi (swarm) päätyy yhteen paikkaan riippumatta matkan varrella tehdyistä ennalta-arvaamattomista liikkeistä [Engelbrecht 2007].

Keinotekoiset immuunijärjestelmät tunnistavat yhteensopivuuksia kuten eläinten valkosolut tuhoavat epäterveitä soluja. Biologisen immuniteetin toiminnasta on esitetty erilaisia teorioita kuten *vaaramalli* (danger theory), jonka mukaan valkosolut hyökkäävät vain vaarallisten solujen kimppuun. [Engelbrecht 2007.] Keinotekoista immuunijärjestelmää on sovellettu esimerkiksi verkoston tietoturvallisuuden parantamiseen oppimalla aiemmista tunkeiluyrityksistä [Antunes and Correia 2006].

Sumeat järjestelmät voivat käyttää epäeksaktia tietoja laskennan suorittamiseen samoin kuin ihmiset voivat tehdä päätöksiä tietämättä kaikkia mahdollisia muuttujia [Engelbrecht 2007]. *Sumea logiikka* (fuzzy logic) pyrkii summittaisen päättelyn formaaliin ilmaisuun [Zadeh 1994]. Esimerkiksi luokitteluissa voidaan hyödyntää *sumeita joukkoja* (fuzzy set), joiden alkiot kuuluvat joukkoon jollakin asteella: pitkien ihmisten joukossa yli kaksimetriset ovat pitkiä suuremmalla asteella tai painoarvolla kuin 1,8-metriset. [Engelbrecht 2007.]

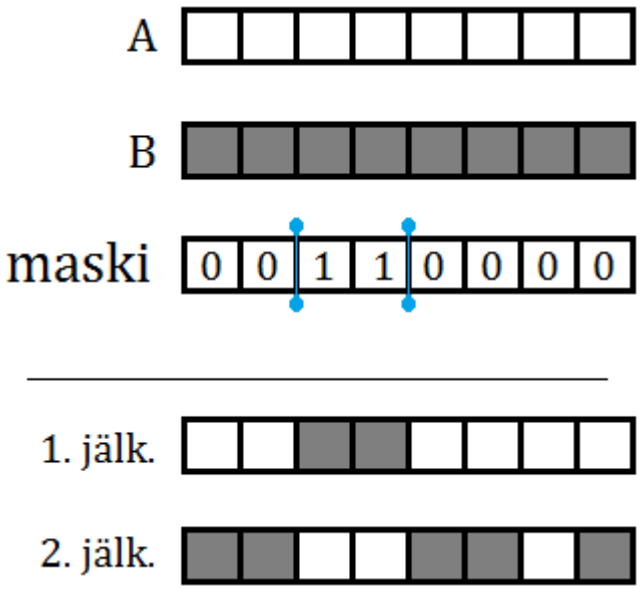
Evoluutiolaskenta perustuu populaation ympäristön suhteen parhaiden selviytymiseen, jolloin vain parhaimmat vanhemmat siirtävät geenejään eteenpäin [Engelbrecht 2007]. Kyseinen piirre yhdistää evoluutiolaskentaa ja harmonista hakua merkittävällä tavalla, joten kuvailen evoluutiolaskentaa muita paradigmoja tarkemmin.

4.2. Evoluutiolaskenta

Evoluutiolaskennalla tarkoitetaan laskennan keinoin simuloituja evoluution toimintamekanismeja kuten luonnonvalinta ja ratkaisujen hyvyys (fitness). Niitä toteuttavat *evoluutioalgoritmit* (evolutionary algorithm, EA), jotka kuvaavat geneerisen mallin luoda algoritmi: sen tulisi alustaa populaatio, esittää ratkaisujoukot (joita kutsutaan kromosomeiksi), arvioida ratkaisujen kykyä selviytyä sekä alustaa valinta- ja lisääntymisoperaattorit. [Engelbrecht 2007.] Geemin [2001] mukaan evoluutioalgoritmeja on neljä: *geneettiset algoritmit* (genetic algorithm, GA), *geneettinen optimointi* (genetic programming), *evolutiivinen optimointi* (evolutionary programming) sekä *evoluutiostrategiat* (evolution strategies, ES).

Geneettiset algoritmit ovat suosituimpia evoluutioalgoritmeja [Yang 2009]. Ne perustuvat yksinkertaisimmillaan kolmeen operaattoriin: *lisääntymiseen* (reproduction), *mutaatioon* (mutation) sekä *ristetykseen* (crossover). Lisääntymään valitaan parhaimmat vanhemmat, ja näiden vanhempien jälkeläisiin syntyy satunnaisia mutaatioita. [Geem et al. 2001.] Risteytys voi tapahtua eri tavoin vanhempien määrästä riippuen: jos risteytys toteutetaan kahdella vanhemmalla bittijonoesitysten avulla, voidaan jälkeläisen bittijonoon valita osia kummankin

vanhemman biteistä [Engelbrecht 2007]. Kuva 2 esittää *kaksipisteistä risteytystä* (two-point crossover), jossa maski ilmaisee, tuleeko ensimmäisen jälkeläisen geeni vanhemmalta A vai B. Geenit jakautuvat kahden pisteen perusteella, jotka ovat kuvan siniset merkit. Toisesta jälkeläisestä tulee ensimmäisen jälkeläisen bittijonon käänteisesitys, johon aiheutettu mutaatio on kääntänyt yhden biteistä uuteen arvoon.



Kuva 2. Risteytys geneettisen algoritmin keinoin. Huomaa toisen jälkeläisen mutaatio.

Geneettinen optimointi on geneettisten algoritmien erikoistuma [Engelbrecht 2007]. Siinä missä geneettiset algoritmit tuottavat evoluution tuloksena entistä parempia ratkaisuja esimerkiksi bitti- tai merkkijonoina, geneettinen optimointi tuottaa kokonaisia tietokoneohjelmia. Ajatuksena on, ettei ratkaisujen kehittyminen riitä, vaan myös itse ohjelman tulee kehittyä evoluutioprosessin tuloksena. [Geem et al. 2001.] Tietokoneohjelmat voidaan esittää puurakenteena, jolloin geneettisen ohjelman kromosomit voivat olla erikokoisia toisin kuin geneettisten algoritmien määrätyn kokoiset jonoesitykset [Engelbrecht 2007].

Evolutiivisen optimoinnin lähtökohtana ovat fenotyypit eli evoluution tuottamien yksilöiden ilmentymät. Tarkemmin ottaen se mallintaa ympäristöön sopeutuvaa käyttäytymistä, jolloin parhaiten ympäristössään operoivat yksilöt selviytyvät. [Engelbrecht 2007.] Geeneihin eli genotyyppiin kohdistuvia muutoksia ei tapahdu kuten esimerkiksi geneettisten algoritmien risteytyksissä. Mutaatiot ovat silti läsnä ja ne ovat evolutiivisen optimoinnin pääasiallinen tapa muuttaa yksilöitä. [Yang 2009.] Engelbrecht [2007] tiivistää, että kyse on kaikkien keskeisestä kilpailusta.

Evoluutiostrategiat ovat algoritmeja, jotka perustuvat kahteen yksinkertaiseen mutaatio- ja valintasääntöön: kaikkia muuttujien arvoja tulee muuttaa pieniä määriä satunnaisesti ja uusi ratkaisu omaksutaan heti, jos se on aiempaa

parempi [Beyer and Schwefel 2002]. Ne eivät käytä risteytystä [Yang 2010]. Alkuperäistä evoluutiostrategiaa kutsutaan (1+1)-variaatioksi [Beyer and Schwefel 2002] ja se kehitettiin järjestelmien automaattiseen parametrien optimointiin [Geem et al. 2001]. Beyerin ja Schwefelin [2002] kuvailemassa esimerkissä kolmiulotteinen kappale kehittää tuulitunnelisimulaatiossa automaattisesti muodon, joka aiheuttaa vähiten ilmanvastusta. Variaation nimitys tulee siitä, että se luo yhdestä vanhemmasta yhden jälkeläisen. Algoritmin myöhemmät versiot laajentavat vähintään toista tekijää: esimerkiksi (μ+1)-variaatio luo yhden jälkeläisen useasta vanhemmasta ja (μ+λ)-versio puolestaan useita jälkeläisiä monesta vanhemmasta. [Beyer and Schwefel 2002.]

4.3. Harmoninen haku algoritmien joukossa

Metaheuristiikat ovat luonnosta tehtyjä mallinnuksia, jotka noudattavat vain muutamaa yleistä suunnitteluperiaatetta. Sen seurauksena metaheuristiikkojen algoritmisilla toteutuksilla on paljon liikkumavaraa; eräs ainoista edellytyksistä on, että ne tuottavat luontaisesti satunnaisuutta. Lisäksi metaheuristiikkoja voi käyttää mitä erilaisimmissa optimointiongelmissa. Nämä seikat tekevät metaheuristiikkojen objektiivisesta arvioinnista monimutkaista. Manjarresin [2013] mukaan ei voida väittää, että jokin metaheuristiikka toimii yleisesti muita paremmin, vaan niitä tulee arvioida keskittymällä sovelluskohteeseen.

Geem [2001] esittelee alkuperäisessä tutkimuksessaan erilaisia optimointiongelmia sekä vertailee harmonisen haun saamia tuloksia muiden metaheuristiikkojen vastaaviin. Hän muun muassa kuvaa minimointitehtävän jatkuva-arvoisella muuttujalla, jolle harmoninen haku tuottaa parhaan rajoitteita noudattavan tuloksen. Vertailu on tehty geneettisiin algoritmeihin ja evolutiiviseen optimointiin, joista molempien laskemat tulokset ovat huonommat eivätkä ne noudata annettuja rajoitteita. Geemin [2009] tutkimuksissa harmoninen haku on järjestään antanut ainakin geneettisiä algoritmeja parempia ratkaisuja.

Manjarres [2013] esittää, että harmonisella haulla on luonnostaan piirteitä, joita hyödyntämällä sen toiminnasta on voitu tuottaa paljon tutkimusta. Ensinnäkin haku käyttää aiempia ratkaisujaan laajasti. Toiseksi ja edelliseen liittyen harmonioilla ei ole määrättyjä vanhempia samalla tavalla kuin geneettisillä algoritmeilla tai evoluutiostrategioilla vaan harmoniamuistia käytetään kokonaisuudessaan, ja mikä tahansa siellä esiintyvä arvo voi sijoittua mihin tahansa muuttujaan uudessa harmoniassa. Kolmanneksi harmonisen haun laskemiin tuloksiin ei vaikuta suuresti annetut lähtöarvot, sillä se ei käytä gradienttia, joka kertoo arvojen muutoksen suuruuden tiettyyn suuntaan. Neljänneksi haku hienosäätää toimintaansa itsenäisesti esimerkiksi kaventamalla värähtelyleveysparametrin tutkimaa aluetta. Lopuksi se on rakenteeltaan yksinkertainen, mikä mahdollistaa joustavan yhdistelyn muihin metaheuristiikkoihin. [Manjarres et al. 2012.]

Yang [2009] kirjoittaa myös syitä harmonisen haun menestykseen ja esittää samankaltaisia syitä kuin Manjarres. Sopeuttamisaste (joka operoi värähtelyleveyden avulla) jalostaa lokaaleja ratkaisuja aina suhteessa valikoituun alkioon. Silloin niiden arvot pysyvät mahdollisimman lähellä jo löydettyjä hyviä ratkaisuja. Lisäksi harmoninen haku ei ole altis tuottamaan huonoja ratkaisuja vain, koska parametreja ei ole tarkasti hienosäädetty. [Yang 2009.]

5. Pohdintaa ja kritiikkiä

Harmoninen haku muistuttaa evoluutioalgoritmeja, mutta sillä on selkeitä eroja niihin nähden. Selvintä on, ettei harmoninen haku tuota ratkaisuja suoritettavina ohjelmina kuten geneettinen optimointi. Lähimpänä lienevät geneettiset algoritmit, jotka tuottavat uusia ratkaisuja risteyttämällä populaation yksilöitä. Vaikka on olemassa muunnelmia geneettisistä algoritmeista, jotka hyödyntävät mielivaltaista määrää vanhempia, tavanomaiset versiot käyttävät kuitenkin vain kahta [Manjarres et al. 2013]. Sen lisäksi uudet ratkaisujoukot muodostetaan pilkkomalla vanhempien genomi pisteiden avulla osiin kuten kuva 2 esittää. Tällöin ratkaisut saavat määrättyjä arvoja määrättyihin kohtiin. Harmonisen haun voi sanoa käyttävän lähtökohtanaan uusia tuloksia, kun geneettiset algoritmit operoivat vanhoista käsin.

Toisaalta harmonisen haun taustalla olevan metaforan mukaisesti uudet harmoniat luodaan korjaamalla vanhojen harmonioiden säveliä. Siinä mielessä harmonioiden improvisointi on ilmiasujen iteratiivista korjaamista. Harmoninen haku muistuttaa siis siltä osin myös evolutiivista optimointia. Nämä algoritmit aiheuttavat mutaatioita yksilöiden fenotyyppiin, ilmiasuun.

Harmoninen haku on kohdannut kritiikkiä sen samanlaisuudesta evoluutiostrategioiden kanssa. Tarkalleen kyse on (μ+1)-variaatiosta, jossa populaation koko μ ei muutu. Samoin harmonisen haun hyödyntämä populaatio tallennetaan harmoniamuistiin, jonka koko on määrätty ja muuttumaton. Uusia ratkaisuja tuotetaan iteratiivisesti yksi kerrallaan, ja ratkaisu tallennetaan, jos se on riittävän hyvä. [Weyland 2010.] Weyland [2010] johtaa tästä, että algoritmin vaiheet ovat identtiset. Hän väittää myös, että uusiin ratkaisujoukkoihin (μ+1)-evoluutiostrategiassa tehdään koko populaation keskeinen risteytys, josta uudet arvot ratkaisujoukkoihin saadaan [Weyland 2010].

Ensinnäkin Beyerin ja Schwefelin [2002] kuvailema (μ+1)-variaatio eroaa merkittävästi harmonisesta hausta samasta syystä kuin geneettiset algoritmit: uusiin ratkaisujoukkoihin käytetään vain kahta aiempaa muistissa esiintyvää ratkaisua. Evoluutiostrategioiden (μ+1)-variaatio, kuten Geem [2010] vastauksessaan kritiikkiin huomauttaa, on siis pikemminkin lähellä geneettisiä algoritmeja. Toisekseen, kuten luvussa 4.2 mainitsin, evoluutiostrategiat eivät käytä risteytystä [Yang 2010].

Weylandin [2010] esittämät väitteet eivät vastaa lukemiani lähteitä etenkään kahden viimeisen väitteen osalta, mutta en voi sulkea pois mahdollisuutta, että (μ+1)-evoluutiostrategiasta on esitetty myös Weylandin kuvaama malli. Hän on käyttänyt lähteenään saksankielistä väitöskirjaa 1970-luvulta, jonka englanninkieliseen käännökseen en päässyt käsiksi. Johtopäätös kuitenkin on, että harmoninen haku on (μ+1)-evoluutiostrategian erityinen muoto ja on siten aina huonompi kuin parhain evoluutiostrategia-algoritmi [Weyland 2012]. Tämä väite on non sequitur: on yhtä validia esittää, että erityinen muoto yleisestä algoritmista on tarkempi ja siten parempi.

6. Yhteenveto

Reaalimaailman monimutkaisiin optimointiongelmiin voidaan löytää ratkaisutapoja, jotka eivät ole perinteistä laskentaa matemaattisilla malleilla. Näitä uusia tapoja voidaan hahmottaa mistä tahansa prosesseista, jotka pyrkivät korjaamaan itseään. Heuristiset mallit ovatkin osoittaneet hyödyllisiksi ja tehokkaiksi keinoiksi, ja 2000-luvun alussa kehitetty harmoninen haku on siitä esimerkki. Se näkee muusikoiden improvisoinnin olevan parhaimpaan harmoniaan pyrkivä optimointiprosessi. Käytännössä harmoninen haku iteroi uusia ratkaisujaan hyödyntämällä kaikkia muistissa olevia harmonioita ja mahdollistaa täten suuren mahdollisten ratkaisujen joukon tutkimisen. Lisäksi harmonioita improvisoidessa voidaan lisätä satunnaisuutta poikkeamalla taulukosta valikoituneista arvoista tai etsimällä ratkaisuja kokonaan sen ulkopuolelta. [Geem et al. 2001.]

Harmonisen haun perusteet eivät tule uutuutena [Geem 2010]: laskennallisen älykkyyden alku on 1950-luvulla ja evoluutiolaskennan kehittäminen alkoi geneettisten algoritmien myötä samoihin aikoihin [Engelbrecht 2007]. Harmoninen haku sisältää paljon elementtejä, joita evoluutioalgoritmit käyttivät kauan ennen harmonisen haun syntyä. Siihen liittyvää tutkimusta on kuitenkin tehty huomattavan paljon eikä samankaltaisuus ole estänyt uusien tuloksien aikaansaamista.

Viiteluettelo

[Antunes and Correia 2006] Mário J. Antunes and Manuel E. Correia. 2006. Towards a new immunity-inspired intrusion detection framework. Technical Report Series: DCC-2006-4. DCC-FC & LIACC, Universidade do Porto.

[Askarzadeh 2013] Alireza Askarzadeh. 2013. A discrete chaotic harmony search-based simulated annealing algorithm for optimum design of PV/wind hybrid system. Solar Energy 97, 93–101.

[Beyer and Schwefel 2002] Hans-Georg Beyer and Hans-Paul Schwefel. 2002. Evolution strategies – a comprehensive introduction. Natural Computing 1, 1, 3–52.

[Blum and Roli 2003] Christian Blum and Andrea Roli. 2003. Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Computing Surveys 35, 3, 268–308.

[Chinneck 2009] John W. Chinneck. 2009. Practical Optimization: A Gentle Intro-duction. Carleton University.

[Dijkstra 1959] E. W. Dijkstra. 1959. A note on two problems in connexion with graphs. Numerische Mathematik 1, 1, 269–271.

[Engelbrecht 2007] Andries P. Engelbrecht. 2007. Computational Intelligence: An Introduction. Wiley.

[Geem et al. 2001] Zong Woo Geem, Joong Hoon and G.V. Loganathan. 2001. A new heuristic optimization algorithm: harmony search. SIMULATION 76, 2, 60–68.

[Geem 2012] Zong Woo Geem. 2012. Research commentary: survival of the fittest algorithm or the novelest algorithm? The existence reason of the harmony search algorithm. In: Peng-Yeng Yin (ed.), Modeling, Analysis, and Ap-plications in Metaheuristic Computing: Advancements and Trends. 84–89.

[Holder 1999] A. Holder. 1999. Mathematical Programming Glossary. INFORMS Computing Society. http://glossary.computing.society.informs.org. Checked 30.10.2015.

[Ingram and Zhang 2009] Gordon Ingram and Tonghua Zhang. 2009. Overview of applications and developments in the harmony search algorithm. In: Zong Woo Geem (ed.), Music-Inspired Harmony Search Algorithm. Springer, 15–37.

[Kyppö 2002] Jorma Kyppö. 2002. Simplex-menetelmä. Jyväskylän yliopisto. http://users.jyu.fi/~jorma/simplex.htm. Tarkistettu 2.11.2015.

[Manjarres et al. 2013] D. Manjarres, I. Landa-Torres, S. Gil-Lopez, J. Del Ser and M.N. Bilbao. 2013. A survey on applications of the harmony search algorithm. Engineering Applications of Artificial Intelligence 26, 1818–1831.

[Mordvintsev et al. 2015] Alexander Mordvintsev, Christopher Olah and Mike Tyka. 2015. DeepDream - a code example for visualizing neural networks. Google Research Blog. http://googleresearch.blogspot.co.uk/2015/07/deepdream-code-example-for-visualizing.html. Checked 28.11.2015.

[NEOS 2013] Network-Enabled Optimization System. 2013. NEOS Optimization Guide. The Morgridge Institute for Research, Inc. http://www.neos-guide.org/Optimization-Guide. Checked 31.10.2015.

[Pandi and Panigrahi 2011] V. Ravikumar Pandi and Bijaya Ketan Panigrahi. 2011. Dynamic economic load dispatch using hybrid swarm intelligence ba-

sed harmony search algorithm. Expert Systems with Applications 38, 7, 8509–8514.

[Snyman 2005] Jan A. Snyman. 2005. Practical Mathematical Optimization: An In-troduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms. Springer.

[Weyland 2012] Dennis Weyland. 2012. A rigorous analysis of the harmony search algorithm: how the research community can be misled by a ”novel” methodology. In: Peng-Yeng Yin (ed.), Modeling, Analysis, and Applications in Metaheuristic Computing: Advancements and Trends. 72–83.

[Yang 2009] Xin-She Yang. 2009. Harmony search as a metaheuristic algorithm. In: Zong Woo Geem (ed.), Music-Inspired Harmony Search Algorithm. Spring-er, 1–14.

[Yang 2010] Xin-She Yang. 2010. Nature-Inspired Metaheuristic Algorithms: Second Edition. Luniver Press.

[Zadeh 1994] Lotfi A. Zadeh. 1994. Fuzzy logic, neural networks, and soft computing. Comm. ACM 37, 3, 77–84.

Pariohjelmoinnin käyttö opetuksessa

Marjut Vornanen

Tiivistelmä.
Pariohjelmoinnissa kaksi ohjelmoijaa työskentelee saman näytön äärellä: toinen kirjoittaa lähdekoodia ja toinen tarkkailee ja esittää kysymyksiä. Pariohjelmointia hyödynnetään ohjelmistoyrityksissä sen tehokkuuden ja koodin laadun paranemisen vuoksi. Pariohjelmoinnista on hyötyä myös ohjelmoinnin opetuksessa. Pariohjelmoinnin myötä oppilaiden arvosanat paranevat ja kurssin kesken jättäneiden määrä vähenee. Lisäksi oppilaiden ohjelmointinopeus kasvaa ja koodin laatu paranee. Oppilaat kokevat vähemmän turhautumisen tunteita eikä koodin kanssa jäädä jumiin yhtä helposti kuin yksin työskennellessä. Opettajalle jää enemmän aikaa auttaa oppilaita. Pariohjelmointi sopii erityisen hyvin tytöille ja naisille, sillä se saa heidät kiinnostumaan ohjelmoinnista ja alan työstä. Tässä tutkielmassa selvitetään, kuinka pariohjelmointia voidaan käyttää ohjelmoinnin opetuksessa.

Avainsanat ja -sanonnat: pariohjelmointi, ohjelmointi, opettaminen, opetus, oppilas, opiskelija.

1. Johdanto

Pariohjelmointi on ollut käytössä jo vuosikymmenien ajan, mutta vasta Extreme Programming -menetelmän myötä se nousi nykyiseen suosioonsa [Beck 2000; Williams and Kessler 2003]. Pariohjelmoinnissa toinen ohjelmoija keskittyy lähdekoodin kirjoittamiseen ja toinen tarkkailee koodia jatkuvasti etsien mahdollisia virheitä ja parempia tapoja kirjoittaa koodia [Preston 2006]. Pariohjelmoinnilla on runsaasti hyviä puolia. Sen on muun muassa todettu parantavan lähdekoodin laatua ja vähentävän testaukseen kuluvaa aikaa. [Bipp *et al.* 2008; Hannay *et al.* 2009; Preston 2006]

Pariohjelmoinnin käyttöä ohjelmoinnin opetuksessa on tutkittu paljon. Sen on koettu tuovan runsaasti hyötyä oppilaille ja myös ohjelmoinnin opettajille. Huonoja puoliakin löytyy, eikä pariohjelmointi välttämättä sovi kaikille. [Preston 2006; Williams and Upchurch 2001] Pariohjelmoinnin tulokset opetuksessa ovat kuitenkin osoittautuneet niin hyviksi, että pariohjelmointia kannattaisi käyttää yhä enemmän ohjelmointikursseilla. Tämä tutkielma selvittää, kuinka ja miksi pariohjelmointia on hyödynnetty ohjelmoinnin opetuksessa.

Toisessa luvussa kerrotaan mitä pariohjelmointi on, minkälaisia erilaisia pariyhdistelmiä on olemassa ja pohditaan yleisellä tasolla pariohjelmoinnin hyötyjä ja ongelmia. Kolmannessa luvussa tutkitaan persoonallisuuden vaikutusta

sekä ohjelmointitaitoihin että yhdessä ohjelmoivien parien yhteensopivuuteen. Neljännessä luvussa tutustutaan pariohjelmoinnin käyttöön opetuksessa. Siinä esitellään pariohjelmoinnin käytön hyviä ja huonoja puolia opetuksessa, kerrotaan pariohjelmoinnin sopivuudesta naisten ja vähemmistöjen opetukseen ja tutustutaan oppilaiden asenteisiin pariohjelmointia kohtaan. Lisäksi kerrotaan, miten oppilaat tulisi jakaa yhteensopiviksi pareiksi, minkälainen on opettajan rooli ja työpanos pariohjelmointia hyödyntävillä ohjelmointikursseilla ja miten pariohjelmointi voidaan toteuttaa etäopiskeluna. Viidennessä luvussa pohditaan pariohjelmoinnin käyttöä opetuksessa yleisellä tasolla. Kuudennessa luvussa on yhteenveto tutkielman sisällöstä.

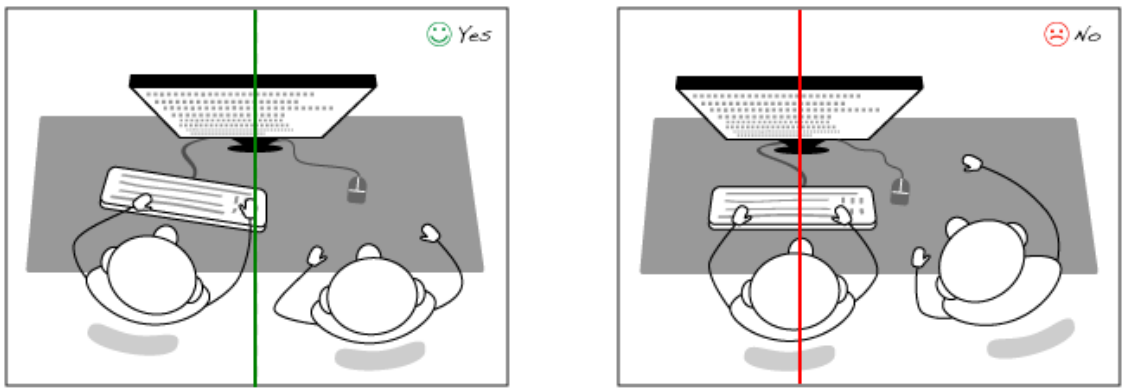
2. Pariohjelmointi

Pariohjelmointia on hyödynnetty hajanaisesti jo vuosikymmenien ajan [Williams and Kessler 2003]. Kuitenkin vasta ketterän ohjelmistokehityksen ja Extreme Programming -metodologian myötä pariohjelmoinnin suosio on kasvanut [Beck 2000]. Extreme Programming (XP) on Kent Beckin 1990-luvun lopulla kehittämä ketterän ohjelmistokehityksen metodologia. Pariohjelmointi on yksi XP-menetelmän 12 käytännöstä. [Williams and Kessler 2003] Pariohjelmoinnista tuli XP:n myötä yhä tunnetumpi 1990-luvun lopussa ja 2000-luvun alussa. Monet epäilivät pariohjelmoinnin toimivuutta. Sen ajateltiin vievän kaksi kertaa enemmän aikaa kuin yksin ohjelmoinnin. Utahin yliopistossa vuonna 1999 tehdyn tutkimuksen positiivisten tulosten ja XP-metodologian yleistymisen ansiosta pariohjelmointia alettiin käyttää yhä enemmän yrityksissä ja oppilaitoksissa. [Williams 2010]

Pariohjelmointi eroaa perinteisestä yksin ohjelmoimisesta eli soolo-ohjelmoinnista siten, että pariohjelmoinnissa kaksi ohjelmoijaa työskentelee samalla työpisteellä. Toinen on ajaja (driver): hän käyttää hiirtä ja näppäimistöä ja kirjoittaa lähdekoodia. Ajajaa voi kutsua myös nimellä kirjoittaja. Toinen on navigaattori (navigator): hän tarkkailee koodia virheiden varalta, pohtii toisia tapoja kirjoittaa koodia ja esittää kysymyksiä parilleen. Navigaattoria sanotaan myös tarkkailijaksi (observer). Parit vaihtelevat tasaisin väliajoin rooleja keskenään. Pariohjelmoinnin aikana tapahtuu soolo-ohjelmointia laajempaa reaaliaikaista laadun kontrollointia, koska ohjelmaa lukee tekijän lisäksi aina toinen henkilö. [Preston 2006]

Pariohjelmointi tulisi ottaa huomioon työtilan järjestelyissä. Kuvassa 1 on esitetty oikealla puolella väärä tapa työskennellä yhdessä saman näytön äärellä: lähdekoodin kirjoittaja on asettunut kohtisuoraan näyttöä kohden ja navigaattori joutuu työskentelemään epäergonomisesti näytön sivussa. Oikea tapa on

esitetty vasemmalla puolella: molemmat ovat hivenen sivussa näytöstä, jolloin molemmat näkevät näytön yhtä hyvin. [Bain 2015]



Kuva 1. Oikea ja väärä tapa työskennellä pareittain yhden näytön äärellä [Bain, 2015].

Pariohjelmointi on *yhteisöllisen oppimisen* (collaborative learning) sovellus, jossa työskennellään yhdessä tavoitteen saavuttamiseksi [Preston 2006].

Williams ja Kessler [2003] ovat havainneet seitsemän pariohjelmointiin liittyvää synergistä ilmiötä:

1. Paripaine. Parityöskentelyssä ihmisillä on tapana työskennellä nopeammin, jotta työkumppanin ei tarvitsisi kokea pettymystä. Pariohjelmointi siis lisää painetta tehdä työtä tehokkaasti ja olla hyvä ohjelmoija.
2. Parineuvottelu. Yhdessä työskennellessä joutuu myös neuvottelemaan keskenään. Aivoriihen onkin todettu olevan tehokas keino ongelmanratkaisuun.
3. Parirohkeus. Parityöskentely myös lisää luottamusta ja rohkaisee. Ohjelmoija oppii tietämään, milloin jokin ratkaisu on oikein ja milloin täytyy myöntää, että on väärässä.
4. Paritarkastelu. Pariohjelmoinnissa on käytössä neljä silmää, jotka tarkkailevat syntyvää lähdekoodia koko ajan. Tämä on hyvä käytäntö aikaiseen vikojen havainnointiin.
5. Parivirheenkorjaus. Tehokas tekniikka vian ratkaisemiseen on selittää koodinpätkä jollekin toiselle. Samalla tulee usein selitettyä vika myös itselleen.
6. Parioppiminen. Heikkoudet ja puutteet ohjelmakoodissa ehkäistään sillä, että tietoa jaetaan jatkuvasti työkumppanien välillä.
7. Pariluottamus. Pareja kannustetaan tuottamaan hyviä ideoita avoimessa ympäristössä.

2.1. Erilaiset pariyhdistelmät

Parien ohjelmointikokemus ja ohjelmointitaidot voivat erota toisistaan tai olla suurin piirtein samat. Williams ja Kessler [2003] esittelivät neljä erilaista pariyhdistelmää:

1. Asiantuntija–asiantuntija.
2. Asiantuntija–keskiverto.
3. Asiantuntija–aloittelija.
4. Aloittelija–aloittelija.

Kahden asiantuntijan työskennellessä yhdessä tulos on erittäin tasokasta [Williams and Kessler 2003]. Kummankaan ei tarvitse opettaa toista ja aikaa säästyy itse ongelman ratkomiseen. Asiantuntijan työskennellessä yhdessä keskiverto-ohjelmoijan kanssa keskinkertainen ohjelmoija pystyy oppimaan asiantuntijalta uusia tapoja ajatella ja ratkaista ongelmia tehokkaammin. Uuden oppiminen vaatii kuitenkin keskiverrolta ohjelmoijalta halua oppia. [Williams and Kessler 2003]

Asiantuntijan ja aloittelijan yhteistyö vaatii asiantuntijalta kärsivällisyyttä, sillä aloittelijalla on paljon opittavaa eikä asiantuntija saa mennä liian nopeasti. Asiat tulee selittää huomattavasti yksityiskohtaisemmin kuin keskiverto-ohjelmoijalle. [Williams and Kessler 2003] Tämä voi tosin olla hankalaa asiantuntijalle, joka on tottunut toimimaan nopeasti. Asiantuntijan voi olla myös hankala asettua aloittelijan asemaan. Hän ei ehkä kykene selittämään asioita tarpeeksi tarkasti. Keskiverto-ohjelmoija osaisi mitä luultavammin opettaa aloittelijaa tehokkaammin. Aloittelija–aloittelija-yhdistelmällä on hyvä ratkoa suhteellisen yksinkertaisia ongelmia. Näin aloittelijat saavat tarvitsemaansa kokemusta. [Williams and Kessler 2003]

Lui ja Chan [2006] huomasivat, että aloittelija–aloittelija-yhdistelmä verrattuna yksin ohjelmoivaan aloittelijaan tuottaa enemmän koodia kuin asiantuntija–asiantuntija-yhdistelmä verrattuna yksin ohjelmoivaan asiantuntijaan. Erityisesti aloittelijoiden kannattaa siis ohjelmoida yhdessä, jotta työn tuottavuus pysyy korkealla.

2.2. Pariohjelmoinnin hyödyt

Pariohjelmoinnin tehokkuutta on pyritty selvittämään useilla tutkimuksilla. Yksinkertaisesti ajateltuna voisi uskoa pariohjelmoinnin olevan soolo-ohjelmointia tehottomampaa. Sen luulisi vievän enemmän aikaa ja lisäävän kustannuksia. Tutkimuksista päätellen voimme kuitenkin todeta pariohjelmoinnilla olevan monia hyviä puolia ja haittojen jäävän vähäisiksi.

Canforan ja kumppaneiden [2005] empiirisessä tutkimuksessa saatiin selville, että ohjelmoijan siirtyessä soolo-ohjelmoinnista pariohjelmointiin ohjelmoin-

titehtävän ratkaisemiseen vaadittu aika pieneni. Myös Hannayn ja kumppaneiden [2009] meta-analyysissa päädyttiin samoihin tuloksiin. Williamsin ja Kesslerin [2003] mukaan ohjelmointiaika jopa puolittuu. Pariohjelmointi on tehokasta etenkin uusia ongelmia [Lui and Chan 2006] ja hyvin monimutkaisia ohjelmointitehtäviä [Hannay *et al.* 2009] ratkaistaessa.

Ajan säästämisen lisäksi pariohjelmointi lisää lähdekoodin laatua ja tuottaa korkealaatuisimpia ohjelmistoja [Hannay *et al.* 2009; Bipp *et al.* 2008; Preston 2006]. Lähdekoodista tulee yksinkertaisempaa, luettavampaa ja helpommin ymmärrettävää kuin yksin ohjelmoidessa. Lähdekoodin ollessa selkeää vähentyy virheiden etsimiseen kuluva aika. [Bipp *et al.* 2008] Näin ollen pariohjelmoinnin hyödyt heijastuvat testausvaiheeseen asti; myös testaamiseen kuluva aika lyhenee. Williamsin ja Kesslerin [2003] mukaan pareittain ohjelmoidessa myös koodiin tulee vähemmän virheitä, joten virheiden korjaamiseenkaan ei kulu niin paljoa aikaa kuin soolo-ohjelmoinnissa. Etenkin monimutkaisia ohjelmistoja kehitettäessä suuri osa ajasta menee virheiden etsimiseen. Testaus ja vikojen etsiminen on helpompaa kahden tiimeissä kuin yksin [Bipp *et al.* 2008].

Bipp ja kumppanit [2008] totesivat, että pareittain työskentely lisäsi tehokkuutta, koska keskittyminen tehtävään oli parempaa. Ongelmatilanteessa ohjelmointikumppani on aina lähellä vastaamassa kysymyksiin. Lisäksi vaihtuvien työkumppaneiden kanssa työskentely lisää työntekijöiden tietoisuutta projektin eri osista. Tällöin kuitenkin työkumppaneita tulisi vaihdella projektin aikana useampaan kertaan.

Pariohjelmointi helpottaa ryhmän vähemmän kokeneiden jäsenten integraatiota ohjelmointitiimiin [Bipp *et al.* 2008]. Pariohjelmoinnin lomassa juuri valmistunut ohjelmoija pääsee heti uudella työpaikallaan tutustumaan paremmin yhteen kokeneempaan ohjelmoijaan. Yhdessä ohjelmoiminen auttaa kokemattomampaa ohjelmoijaa ratkaisemaan uusia ongelmia. Kokenut ohjelmoija jakaa tietojaan ja taitojaan. Ohjelmakin pysyy laadukkaana, kun sitä ei tee kokonaan kokemattomampi kehittäjä. Lisäksi ohjelmointi pareittain lisää ohjelmointiprosessin ymmärrystä työryhmässä [Preston 2006].

Pariohjelmoinnista on paljon sosiaalista hyötyä. Se lisää oppimista, sillä parit oppivat jatkuvasti seuraamalla toista pariaan. Parin jäsenten välillä siirtyy tieto ja taitoa molempiin suuntiin. Pariohjelmoijat tutustuvat ryhmänsä jäseniin paremmin, mikä lisää luottamusta ja parantaa tiimityötä. Lisäksi pariohjelmoijat ovat yleensä onnellisempia kuin yksin ohjelmoivat kehittäjät. Tämä lisää tyytyväisyyttä työpaikkaan ja henkilöstön liikkuvuus vähenee. [Williams and Kessler 2003] Pariohjelmointi myös tekee ohjelmoinnista nautittavampaa [Preston 2006].

2.3. Pariohjelmoinnin ongelmat

Tutkimuksien tulokset pariohjelmoinnista ovat olleet pääosin erittäin positiivisia. Pariohjelmoinnilla vaikuttaa olevan runsaasti hyötyjä verrattuna yksin ohjelmoimiseen. Eräs esille tullut huono puoli on ollut se, että pariohjelmointi vie jonkin verran enemmän aikaa ja sen myötä menetetään tehokkuutta [Bipp *et al.* 2008]. Nämä haitat kuitenkin korvataan koodin laadussa ja lyhyempänä testausvaiheena. Suurimmassa osassa tutkimuksista pariohjelmoinnin ei kuitenkaan ole huomattu vievän enemmän aikaa tai vähentävän tehokkuutta [Hannay *et al.* 2009]. Eri tutkimuksista on siis saatu ristiriitaisia tuloksia liittyen pariohjelmoinnin kuluttamaan aikaan.

Hannay kumppaneineen [2009] totesi meta-analyysissaan taipumusta pariohjelmointiin liittyvien tutkimusjulkaisujen puolueellisuuteen. On siis mahdollista, että pariohjelmoinnista halutaan antaa parempi kuva ja julkaista vain ne tutkimukset, joista on saatu aikaan hyviä tuloksia. Stephens ja Rosenberg [2003] mainitsevat pariohjelmoinnin olevan vähiten hyödyllisin XP-menetelmän 12 käytännöstä.

3. Pariohjelmointi ja persoonallisuus

Ohjelmointi vaatii kykyä keskittyä ongelman ratkaisemiseen. Stereotyyppisesti ohjelmointia tehdään yksin omien ajatusten keskellä. Siinä missä luonteeltaan introvertti ohjelmoija nauttii ongelmien ratkaisemisesta itsenäisesti omassa rauhassa, ekstrovertti vaatisi työpäivältään enemmän virikettä ja kanssakäymistä muiden ihmisten kanssa. Pariohjelmointi voisi tarjota tällaista virikkeellisyyttä ekstrovertille, mutta samalla se voi olla liikaa omaa rauhaa kaipaavalle introvertille.

Big Five -näkemyksen mukaan on olemassa viisi suurta persoonallisuuspiirrettä: ekstroversio, neuroottisuus, tunnollisuus, sovinnollisuus ja avoimuus. Useat tutkimukset ovat vahvistaneet näiden piirteiden olemassaolon. Ekstrovertti on ulospäin suuntautunut ihminen, joka saa energiansa sosiaalisesta kanssakäymisestä. Ekstrovertin vastakohta on introvertti, jota liika sosiaalinen kanssakäyminen väsyttää. Introvertti saa energiansa yksinolosta. Neuroottinen ihminen on stressaantuu ja huolestuu helposti. Tunnollinen ihminen on hyvin järjestelmällinen ja itseohjautuva. Sovinnollinen ihminen on luotettava ja empaattinen. Avoimuus kuvastaa avoimuutta uusille kokemuksille ja uusille ajattelutavoille. Avoin ihminen on luova ja mielikuvituksellinen. [Nettle 2007]

Kuvasta 2 voi havaita, mitkä ominaisuudet kuvaavat kutakin persoonallisuuspiirrettä vahvasti (korkeat pisteet) ja mitkä ominaisuudet eivät ole ominaisia kyseiselle persoonallisuuspiirteelle (matalat pisteet).

Big Five -persoonallisuuspiirteet		
Tyyppi	**Korkeat pisteet**	**Matalat pisteet**
Ekstroversio	Ulospäin suuntautunut, innostunut	Etäinen, hiljainen
Neuroottisuus	Altis stressille ja huolestumiselle	Emotionaalisesti vakaa
Tunnollisuus	Järjestäytynyt, itseohjautuva	Spontaaninen, huolimaton
Sovinnollisuus	Luottavainen, empaattinen	Yhteistyöhaluton, vihamielinen
Avoimuus	Luova, mielikuvituksellinen, eksentrinen	Käytännöllinen, tavanomainen

Kuva 2. Big Five -persoonallisuuspiirteiden ominaisuudet [Nettle 2007].

Gnambs [2015] tutki meta-analyysimenetelmää käyttäen, kuinka yksilölliset eroavaisuudet persoonallisuudessa ennustavat soveltuvuutta ohjelmoijaksi. Big Five -mallin viidestä persoonallisuuspiirteestä avoimuus, tunnollisuus ja introversio selittivät soveltuvuutta. Sovinnollisuudella ja neuroottisuudella ei ollut yhteyttä soveltuvuuden kanssa. Suurimmaksi selittäväksi selittäjäksi nousi introversio.

Gnambs [2015] pyrki selittämään tuloksia Eysenckin neurobiologisella persoonallisuusteorialla. Introvertit tarvitsevat vain vähän stimulaatiota ympäristöltään. Liiat ärsykkeet voivat saada introvertin väsyneeksi. Ohjelmistokehityksen tehtävät vaativat suunnittelumallien ja -konseptien syvää analysointia. Introvertti soveltuu siis näihin tehtäviin hyvin verrattuna ekstroverttiin. Ekstrovertit voivat tuntea olonsa alivirittyneeksi ohjelmointitehtävien aikana, koska heillä on suurempi tarve olla sosiaalinen muiden ihmisten kanssa.

Salleh kumppaneineen [2010] tutki neuroottisuuden vaikutusta pariohjelmointiin. Saatujen tuloksien mukaan neuroottisuus ei vaikuta mitenkään erityisesti suoritukseen pariohjelmoinnissa. Tämä täsmäisi myös Gnambsin [2015] tuloksiin, joiden mukaan neuroottisuus ei vaikuta ohjelmointisoveltuvuuteen. Sen sijaan ne tiimijäsenet, joiden tunnollisuuden taso oli korkea tai samantasoinen, saavuttivat ohjelmoidessaan parempia tuloksia.

Tulosten mukaan pariohjelmoinnissa tulisi siis keskittyä siihen, että ohjelmoijat olisivat yhtä tunnollisia. Ohjelmoijan itsessään tulisi kuitenkin olla sekä avoin uusille ideoille, tunnollinen ja luonteeltaan introvertti. Tutkimuksissa ei kuitenkaan selvitetty, soveltuuko pariohjelmointi introverteille vai vaatiiko se ulospäin suuntautuneisuutta.

4. Pariohjelmoinnin käyttö opetuksessa

Pariohjelmointia voi hyödyntää myös ohjelmoinnin opettamisessa peruskoulusta yliopistoihin asti. Pariohjelmointi on todettu erittäin päteväksi opetusmenetelmäksi ohjelmointikursseilla ja sitä voisi hyödyntää nykyistä enemmän ohjelmoinnin opetuksessa.

Ohjelmointikursseilla tulisi vaatia oppilaita myös kirjoittamaan raportteja. VanDeGrift [2004] tutki pariohjelmoinnin ja kirjoittamisen yhdistämistä. Tutkimuksessa saatiin selville, että pariohjelmointiprojekteihin liitetty kirjoitusprosessi raportin muodossa varmisti, että parin molemmat osapuolet ymmärsivät projektin ratkaisut. Kirjoittamalla oppilaat pystyivät selittämään käyttämänsä ratkaisut, mikä lisäsi oppilaiden ymmärrystä lopputuloksesta. Raporttien avulla pystytään myös keräämään tietoa oppilaiden käyttämistä prosesseista, heidän kohtaamistaan hankaluuksista ohjelmointiprojektin aikana ja heidän käyttämistään tukiresursseista.

4.1. Pariohjelmoinnin hyödyt opetuksessa

Pariohjelmoinnin käyttö opetuksessa on tuottanut erinomaisia tuloksia. Oppilaiden koearvosanat paranivat ja kurssin suorittaneiden määrä nousi, kun yhä harvempi lopetti kurssin kesken. [Preston 2006] Oppilaat oppivat nopeammin. Ohjelmointinopeus kasvoi ja koodin laatu parani. [Williams and Upchurch 2001] Lisäksi oppilaiden luottamus omaan työhönsä kasvoi [Hanks 2005].

Pariohjelmoinnilla on myös runsaasti sosiaalisia hyötyjä. Pariohjelmointi vaatii keskustelua parin kanssa ja ideoiden miettimistä yhdessä. Ohjelmoidessa parin kanssa kommunikointi lisääntyy, oppilaat ovat onnellisempia eivätkä turhaudu niin helposti [Williams and Upchurch 2001]. Yksin ohjelmointiongelman ratkaiseminen voi tuntua mahdottomalta, mutta parin kanssa ongelman ratkaisu on tehokkaampaa. Pari voi myös tarjota henkistä tukea. Oppilas ei enää helposti tunne olevansa huono ohjelmoija. Ohjelmoinnista enemmän ymmärtävä pari voi tarjota heikommalle osapuolelle tarvittavaa apua. Ongelma voi avautua helpommin sanallisesti kuin oppikirjasta lukemalla. Umar ja Hui [2012] havaitsivat pariohjelmoinnin kannustavan verbaalisia oppilaita puhumaan parinsa kanssa, koska tällaiset oppilaat oppivat parhaiten muuntamalla ajatuksensa sanoiksi.

Williams ja Upchurch [2001] havaitsivat, että oppilaat eivät enää kokeneet niin suurta houkutusta huijata kokeessa. Ero aiempaan oli merkittävä. Syinä tähän olivat mitä todennäköisemmin aiemmin mainitut seikat eli oppilaat ylipäänsä oppivat paremmin eivätkä täten edes kokeneet tarvetta huijata, koska kokeesta pääsisi läpi omilla taidoilla.

Pariohjelmointi helpottaa myös kurssin opettajan työtä, sillä pariohjelmoinnin hyödyntäminen ohjelmointikurssilla vähentää oppilaiden riippuvuutta opettajasta, mikä vähentää opettajien työmäärää [Preston 2006; Williams and Upchurch 2001]. Vaikka opettajan avulle on tarvetta aina toisinaan, osa ongelmista voidaan kuitenkin saada ratkaistua oppilaiden yhteisen pohdinnan ja keskustelun voimin.

4.2. Naiset ja vähemmistöt

Pariohjelmoinnin on todettu olevan hyödyllistä kaikille oppilaille [Werner *et al.* 2004], mutta erityisen hyödyllinen opetustyyli naisille [Preston 2006] sekä vähemmistöille [Katira *et al.* 2005]. Miehille hyöty on vähäisempi [Hanks 2006]. Werner kumppaneineen [2004] on tutkinut pariohjelmoinnin vaikutusta IT-alalla opiskeleviin naisiin. Heidän mukaansa pariohjelmointi on hyödyllistä naisille siksi, että pariohjelmoinnin yhteisöllinen luonne saa naispuoliset oppilaat huomaamaan, että sovelluskehitys ei ole kilpailullista, eikä se eristä sosiaalisesti muista ihmisistä. Ohjelmointi yhdessä saa naiset näkemään tietojenkäsittelytieteen mahdollisena opiskelualana ja urana.

IT-ala ja ohjelmointi on nähty perinteisesti hyvin miehisenä alana, mitä se toki yhä nykypäivänäkin on. Ohjelmointityö nähdään yksinäisenä puurtamisena. Koska monille naisille sosiaalinen kanssakäyminen on tärkeää, on hyvä saada naiset huomaamaan, että ohjelmoiminen voi olla myös sosiaalista työtä yhdessä työkavereiden kanssa. Pariohjelmoinnin on todettu olevan tehokas tapa saada naiset kiinnostumaan ohjelmoinnista sekä lisätä heidän menestystään alalla [McDowell *et al.* 2006].

4.3. Pariohjelmoinnin haitat opetuksessa ja oppilaiden asenteet pariohjelmointia kohtaan

Pariohjelmoinnin huonoksi puoleksi oppilaat ovat kokeneet aikataulujen yhteensovittamisen. Lisäksi on koettu, etteivät he ymmärrä ohjelmakoodiaan niin hyvin kuin yksin ohjelmoidessa. Oppilaiden tunne saavutuksesta oli pienempi ohjelmoidessa parin kanssa kuin yksin. [Simon and Hanks 2008] Oppitunneilla järjestettävä pariohjelmointi voi myös aiheuttaa äänekkään ilmapiirin, joka voi vaikeuttaa oppilaiden keskittymistä ja oman parin äänen kuulemista.

Oppilaiden asenteet pariohjelmointia kohtaan ovat olleet positiivisia. Oppilaat pitävät pariohjelmointia hyödyllisenä oppimista ajatellen, ja oppilaat kokevat saavansa runsaasti tukea pariltaan [VanDeGrift 2004]. Simon ja Hanks [2008] totesivat, että oppilaat kokivat jäävänsä vähemmän jumiin lähdekoodin kanssa pariohjelmoinnissa kuin yksin ohjelmoidessa. Pariohjelmointi lisäsi ajatusten ja mahdollisten ratkaisujen tutkimista. Pariohjelmoinnin koettiin olevan hyvä keino tutustua muihin kurssin oppilaisiin. Pariohjelmointi koettiin yleisesti hyväksi tavaksi oppia ohjelmoimaan.

Hanks [2006] tutki oppilaiden asennetta pariohjelmointia kohtaan. Tutkimuksessa selvisi, että kurssin ohjaaja saattaa vaikuttaa oppilaiden asenteisiin pariohjelmointia kohtaan. Yhdellä tutkitulla ohjelmointikurssilla asenteet olivat positiivisemmat, kun taas toisella eri ohjaajan kurssilla asenteet olivat selkeästi vähemmän positiiviset. Koska pariohjelmointi on hyödyllisempää naisille kuin

miehille, kurssilla ollut ohjaaja vaikutti erityisesti naisten asenteisiin kurssilla. Eroina eri ohjaajien kursseilla ovat ainakin ohjaajan oma persoona ja ohjaajan käyttämät opetustavat. Myös ohjaajan asenne oppilaisiin voi hyvinkin vaikuttaa asenteisiin. Ohjaaja saattaa tiedostamattomasti tai tietoisesti jakaa oppilaat eri arvoryhmiin, joka vaikuttaa hänen asenteisiinsa oppilaitaan kohtaan.

4.4. Opiskelijaparien yhteensopivuus

Katira kumppaneineen [2005] tutki, minkälaiset parit toimivat tehokkaimmin pariohjelmoinnissa. Tutkimus suoritettiin Yhdysvalloissa Pohjois-Carolinan yliopistossa. Tutkimuksessa todettiin, että ohjelmistotuotantoa opiskelevat kokivat yhteenkuuluvuutta eniten sellaisen kumppanin kanssa, jolla on suurin piirtein samanlaiset arvosanat. Yhteenkuuluvuutta lisäsi myös sama sukupuoli. Myöskin vähemmistöryhmiä edustavat oppilaat tunsivat enemmän yhteenkuuluvuutta toisen vähemmistöryhmään kuuluvan oppilaan kanssa. Sen sijaan oppilaiden persoonallisuudella ja itsetunnolla ei ollut vaikutusta yhteensopivuuteen.

Parhaat tulokset pariohjelmoinnilla opetuksessa saadaan siis, kun parin molemmat jäsenet ovat yhtä motivoituneita opiskeluun ja menestyvät opinnoissa yhtä hyvin. Parin tulee olla samaa sukupuolta ja vähemmistöryhmien edustajat tulisi yhdistää pareiksi. Ongelmaksi jääkin se, tulisiko ohjelmointikurssin ohjaajan yhdistää oppilaat pareiksi vai pitäisikö oppilaiden itse antaa valita oma pari, joka ei kuitenkaan välttämättä olisi se kaikista yhteensopivin. Ohjaajalle ei myöskään välttämättä ole mahdollista katsoa oppilaiden keskiarvoja, eivätkä kaikki oppilaat halua ilmoittaa omaa keskiarvoaan ääneen koko luokalle, joten valintaperusteiksi jää enää vain sukupuoli ja vähemmistöryhmiin kuuluvuus.

4.5. Pariohjelmoinnin toteuttaminen etätyönä

Pariohjelmointi perinteisessä muodossaan edellyttää kehittäjien olevan samassa paikassa samaan aikaan. Oppilaiden voi kuitenkin olla hankala sovittaa lukujärjestyksiään yhteen, jotta he voisivat ohjelmoida pareittain. Hanksin [2005] mukaan tämä voi olla yhtenä syynä siihen, miksi monet opettajat eivät hyödynnä pariohjelmointia ohjelmointikursseillaan, vaikka kyseinen menetelmä luo selkeitä hyötyjä oppimiseen.

Hanks [2005] vertaili samassa paikassa ohjelmoivia oppilaita oppilaisiin, jotka suorittivat pariohjelmoimisen eri paikoissa (distributed pair programming). Tutkimuksessa ei löydetty merkittäviä eroja näiden ryhmien välillä. Voidaan siis olettaa, että oppilaat voivat aivan hyvin myös ohjelmoida pareittain, vaikka he eivät fyysisesti tapaakaan toisiaan saman tietokoneen äärellä.

Pariohjelmointi toteutetaan yleensä jakamalla lähdekoodia kirjoittavan ruutu navigaattorina toimivalle parille (remote pair programming). Tämä voidaan toteuttaa etätyötilanteessa käyttämällä esimerkiksi seuraavia ohjelmistoja: Virtual Network Computing, Adobe Connect tai Skype. Navigaattorina toimiva pari pystyy tällöin näkemään omalta näytöltänsä parinsa kirjoittaman lähdekoodin reaaliajassa. Riippuen ohjelmistosta myös parin hiiri saattaa näkyä jaetussa ruutukuvassa. [Schenk *et al.* 2014] Webkameran kautta oppilaat pystyvät halutessaan näkemään toisensa ja ääni kulkeutuu tietokoneen mikrofonin kautta.

Toinen tapa etätyöskentelyyn on käyttää jaettua ohjelmointiympäristöä, jolloin molemmilla ohjelmoijilla on kopio tiedostosta koneellaan ja tiedostoa synkronoidaan jatkuvasti. Tällöin molemmat pystyvät muokkaamaan lähdekoodia tarvittaessa. Hyvänä puolena on myös se, että molemmat näkevät kursorin liikkeet ja tekstieditorin näkymän muutokset (esimerkiksi tiedoston vaihtumisen ja vierittämisen). Lisäksi molemmat voivat käyttää haluamiaan asetuksia ohjelmointiympäristössä. Tätä työskentelytapaa tukevia työkaluja ovat muun muassa Sangam ja XPairtise, jonka saa asennettua lisäosaksi Eclipse-ohjelmistoympäristöön. [Schenk *et al.* 2014]

5. Pohdinta

Pariohjelmoinnin käytöllä opetuksessa on runsaasti hyötyjä ja vain vähän haittoja. Sitä kannattaisi siis mielestäni hyödyntää yhä enemmän ohjelmoinnin opetuksessa kouluissa. Tietojenkäsittelytieteen koulutusohjelmaan tai muuhun vastaavaan hakeutuneet nuoret ovat usein kiinnostuneita ohjelmoinnista, ja heillä on motivaatiota opetella uusi taito. Kotitehtävien tekeminen ei ole motivoituneelle oppilaalle vaativaa ja uusien ongelmien ratkominen koetaan kiinnostavaksi. Niille, joille ohjelmointikurssi onkin pakollinen opinto-ohjelmaan kuuluva kurssi eikä omaa kiinnostusta ohjelmointiin ole, voi kurssin läpikäyminen olla ikävää ja aikaa vievää – etenkin jos tunneilla keskittymisestä huolimatta opetettuja asioita ei kykene helposti ymmärtämään. Tämä voi tosin olla ongelma myös niille tietojenkäsittelytiedettä opiskeleville, jotka eivät koe intoa ohjelmointiin eivätkä halua päätyä ohjelmoijan ammattiin tai ohjelmointi koetaan hankalaksi. Yhdellä ohjelmointikurssilla voi siis olla monenlaisia oppijoita ja motivaation aste voi vaihdella. Tämä tulisi ottaa huomioon opetuksessa.

Opettajalla on usein vain yksi tyyli opettaa ja tämän tyylin tulisi palvella kaikkia kurssilla mukana olevia oppilaita. Jotkut oppilaat oppivat hyvin ohjelmointia itsenäisestikin ohjelmointikirjojen tai internetin avulla, mutta toiset kaipaavat enemmän ohjausta oppiakseen. Aina opettajan tarjoamat selitykset eivät saa oppilasta ymmärtämään. Pariohjelmointi tarjoaa oppilaille mahdolli-

suuden keskustella muiden oppilaiden kanssa. Siinä missä opettajan esitys ei ehkä avannut asiaa tarpeeksi, vastikään itsekin asian oppineen ohjelmointikumppanin selitys saattaa tarjota juuri sen näkökulman, joka saa oppilaan ymmärtämään opetetun asian. Oppilaille saattaa myös olla helpompaa avautua luokkatoverille kuin opettajalle, jos hän ei ymmärrä jotakin.

Pariohjelmointi myös vähentää opettajan työtä [Preston 2006; Williams and Upchurch 2001]. Pariohjelmoinnissa autettavien määrä vähenee puolella. Myös oppilaiden odotusaika vähenee, kun jokainen ei tee tehtäviä yksin vaan pareittain, ja näin ollen opettaja ehtii nopeammin neuvomaan seuraavaa apua tarvitsevaa paria. Monet ongelmat tulevat ratkaistuksi jo pelkästään parien välisellä keskustelulla ja pohdinnalla eikä opettajaa tarvita niin usein avuksi.

Tunnilla suoritettu pariohjelmointi voi aiheuttaa luokkaan melua, joka voi haitata niiden työsuoritusta, jotka eivät pysty keskittymään liiassa metelissä [Williams and Kessler 2003]. Myös vierustoverin äänen kuuleminen voi osoittautua hankalaksi, jonka vuoksi työskentely häiriintyy ja hankaloituu. Kaikki luokkatilat eivät myöskään välttämättä sovellu hyvin pariohjelmointiin. Mikäli pariohjelmointia haluaisi hyödyntää tehokkaasti, kannattaisi luokasta muodostaa pariohjelmoinnille otollinen ympäristö, joka ottaisi huomioon mahdollisen melun ja kahden ihmisen työskentelyn saman päätteen äärellä.

Pariohjelmoinnin hyödyntämisessä tehokkaasti on otettava huomioon, miten oppilaat jaetaan pareiksi. Jos oppilaat saavat itse valita, he todennäköisesti valitsevat tutun ihmisen, esimerkiksi parhaan kaverinsa. Tästä voi aiheutua sekä haittaa että hyötyä. Kaverin lähdekoodia uskalletaan ehkä helpommin arvostella ja keskustelu on vapaampaa. Toisaalta puheenaiheet saattavat helposti lipsua pois ohjelmoinnista [Williams and Kessler 2003]. Mikäli opettaja tuntee oppilaansa hyvin, on hänen helpompaa jakaa oppilaat pareiksi. Usein etenkin yliopistoissa opettaja ei kuitenkaan tunne kaikkia oppilaitaan, vaan parien muodostaminen tapahtuisi tutkimuksiin perustuvien tulosten mukaan eli muun muassa sukupuolen mukaan. Koska ohjelmointikurssille ilmoittautuneita voi olla satoja, kätevintä olisi muodostaa parit jonkinlaisen automaattisen järjestelmän avulla. Oppilailla voisi olla mahdollisuus vaihtaa paria, mikäli järjestelmä on esimerkiksi ehdottanut pariksi oppilasta, jonka kanssa ei tule toimeen.

Etenkin yksin ohjelmoimaan tottuneille ajatus pareittain työskentelystä voi tuntua hankalalta. Myös introvertit saattavat kokea pariohjelmoinnin sosiaalisen puolen liikaa voimia vievänä. [Williams and Kessler 2003] Ohjelmointikurssin opettajan tulisikin harkita, onko parityöskentely pakollista kaikille vai saisivatko oppilaat päättää itse kumman tavan työskennellä he haluavat valita. Tässä tilanteessa soolo-ohjelmoinnin valinnut olisi kuitenkin voinut hyötyä enem-

män pariohjelmoinnista. Koska pariohjelmointi lisää kurssin hyväksytysti suorittaneiden määrää ja vieläpä paremmin arvosanoin [Preston 2006], kannattaisi pariohjelmoinnin olla pakollista etenkin kaikille ohjelmoinnin alkeiskurssin osallistujille. Näin ollen oppilaat todella oppivat paremmin ohjelmoimaan ja suurin osa saa kurssin suoritettua.

Pariohjelmoinnissa on kuitenkin se vaara, että parempi ohjelmoija kirjoittaa lähdekoodia ja huonompi tyytyy katselemaan sivusta eikä kykene tarjoamaan pohdinta-apua. Tässä tapauksessa parin kokemattomampi osapuoli ei ehkä edes ymmärrä parinsa aikaansaamaa koodia. Tämän vuoksi olisikin mielestäni tärkeää, että oppilaat joutuisivat myös kirjoittamaan raportin ohjelmointitehtävistään kuvaten raportissa käyttämiään menetelmiä ongelmien ratkaisemiksi. Tämä keino auttaisi kokemattomampaa ohjelmoijaa ymmärtämään lähdekoodia paremmin [VanDeGrift 2004]. Kun kokemattomampi osapuoli on koodin kirjoittajana ja kirjoittaa jatkuvasti syntaksiltaan väärää tai huonoa koodia, voi kokeneempi turhautua joutuessaan jatkuvasti pyytämään korjaamaan koodia. Kokenut osapuoli voi jopa kokea ikävänä velvollisuutena löytää koko ajan jotakin korjattavaa toisen koodista. Olisikin erittäin tärkeää, että parin molemmat osapuolet ovat suurin piirtein samantasoisia ohjelmointitaidoissaan sekä kyvyssään oppia uutta. Koska parien muodostaminen järkevästi voi olla erittäin hankalaa, kannattaisi parit jakaa uudelleen joka tapaamiskerta. Oppilaat saisivat erilaisia kokemuksia työskennellessään erilaisten ihmisten kanssa. He näkisivät eri ohjelmointityylejä ja oppisivat työskentelemään eri tavoin.

Pariohjelmoinnin myötä tulee ongelma aikatauluttamisesta [Simon and Hanks 2008]. Mielestäni hyvä keino tähän on lisätä kurssille oma aika, joka on varattu kotitehtävien tekemiseen. Esimerkiksi 1,5 tuntia kestävän opetuksen jälkeen voisi olla 1,5 tuntia kestävä aika varattuna samassa luokassa kotitehtäviä varten. Oppilaiden ei tarvitsisi varata omaa aikaa kotitehtävien tekemiseen, vaan sitä varten olisi jo valmiiksi lukujärjestyksessä aika. Näin ollen tuon ajan tulisi sopia kaikille kurssille ilmoittautuneille, eikä ongelmaa parin kanssa yhteisen ajan järjestämisestä olisi. Vaikka etätyöskentelykin on mahdollista, voi samassa paikassa työskentely tuntua miellyttävämmältä.

Pariohjelmoinnin on todettu olevan tehokas opetustapa etenkin naisille. Naisille sosiaalisuus on keskimäärin tärkeämpää kuin miehille [Slaten *et al.* 2005]. Siinä missä miehiä kiinnostaa enemmän selvittää, miten jokin laite toimii, naisille kiinnostus tietojenkäsittelyä kohtaan liittyy enemmän laitteen sosiaaliseen käyttöön eikä laitteeseen itsessään [Margolis *et al.* 1999]. Pariohjelmointia kannattaisikin käyttää etenkin peruskoulussa ja lukiossa, jotta tytöt kiinnostuisivat ohjelmoinnista ja pyrkisivät yhä enemmän IT-alalle. Tällä hetkellä ala on yhä hyvin miesvaltainen. Naiset näkevät ohjelmoimisen yksinäisenä puurtami-

sena. Koska tekniikan kehittyessä perinteiset suorittavat työt tulevat vähenemään, kannattaisi naisiakin saada IT-alalle. Myös naisten näkökulmaa tarvitaan suunniteltaessa ohjelmistoja.

Tyttöjä ja naisia kiinnostaa peleissä yhdessä työskentely jonkin päämäärän saavuttamiseksi eikä niinkään toisen voittaminen. Sosiaalinen vuorovaikutus pelin hahmojen ja muiden pelaajien välillä koetaan myös kiinnostavaksi. Tytöt ratkaisevat mieluummin pulmallisia tehtäviä kuin pelaavat pelejä, jotka vaativat silmän ja käden yhteistyöskentelyä. Tytöt myös usein samaistuvat pelien hahmoihin. [Gorriz and Medina 2000] Mielestäni nämäkin tulokset tukevat tyttöjen ja naisten mieltymystä pariohjelmointiin.

6. Yhteenveto

Pariohjelmoinnin suosio kasvoi 1990-luvun lopussa etenkin ketterän ohjelmistokehityksen Extreme Programming -metodologian suosion ja pariohjelmoinnista saatujen hyvien tutkimustulosten myötä. Pariohjelmoinnissa kaksi ohjelmoijaa työskentelee yhdessä saman päätteen äärellä: toinen toimii lähdekoodin kirjoittajana ja toinen katselmoi koodia vierestä.

Pariohjelmoinnilla on monia hyviä puolia verrattuna perinteiseen soolo-ohjelmointiin. Tutkimusten mukaan koodin kirjoittamiseen kuluva aika ei kaksinkertaistu vaan saattaa jopa vähentyä. Lähdekoodista tulee laadukkaampaa, luettavampaa ja ymmärrettävämpää. Lisäksi se sisältää vähemmän virheitä, jolloin testausvaiheessa virheiden etsimiseen kuluva aika pienenee ja ohjelmistoista tulee korkealaatuisia. Ohjelmoijien keskittyminen on parempaa tiimeissä ja työskentelystä tulee täten tehokkaampaa. Pariohjelmoinnin myötä uudet jäsenet integroituvat tiimiin helpommin. Kokemattomampi ohjelmoija oppii kokeneemman ohjelmoijan kanssa hyviä ohjelmointitapoja. Tietoa ja taitoa kulkee molempiin suuntiin.

Persoonallisuudella on havaittu olevan vaikutusta siihen, kenelle ohjelmointi sopii. Hyvä ohjelmoija on avoin, tunnollinen ja introvertti. Big Five -piirteistä vain neuroottisuudella ja sovinnaisuudella ei ole havaittu olevan yhteyksiä ohjelmointitaitoihin. Persoonallisuus vaikuttaa myös parityöskentelyyn. Hyvän ohjelmointiparin molemmat jäsenet on yhtä tunnollisia. Erot tunnollisuudessa vähentävät parien yhteensopivuutta.

Pariohjelmointi tuo hyötyjä myös ohjelmoinnin opetukseen. Pariohjelmoinnin käyttö opetuksessa paransi oppilaiden arvosanoja ja vähensi kurssin suorittamatta jättäneiden määrää, eivätkä he kokeneet enää yhtä paljon houkutusta huijata tentissä. Oppilaat kokivat luottamuksensa kasvaneen omaa työtään kohtaan. Ohjelmointinopeus kasvoi ja koodin laatu oli parempaa. Oppilaat kokivat vähemmän turhautumisen tunteita, jäivät vähemmän jumiin koodin

kanssa ja olivat onnellisempia. Muut oppilaat tarjosivat henkistä tukea. Naisille ja vähemmistöryhmille pariohjelmointi on erityisen hedelmällistä. Naiset näkevät ohjelmoinnin sosiaalisen puolen ja kiinnostuvat ohjelmoinnissa ja alan työstä. Pariohjelmoinnin myötä opettajan työ vähenee, kun osa ongelmista kyetään ratkaisemaan parien välisen ajatusten vaihdon kautta. Oppilaiden asenne pariohjelmointia kohtaan on myönteistä, ja he kokevat pariohjelmoinnin hyödylliseksi.

Pariohjelmoinnilla on havaittu myös olevan muutamia huonoja puolia opetuksessa. Oppilaiden saattaa olla vaikea sovittaa aikataulujaan yhteen, jotta he voisivat tehdä yhdessä ohjelmointikotitehtävät. Myös saavutuksen tunne voi vähentyä, ja osa oppilaista ei ehkä ymmärrä kirjoittamaansa lähdekoodia. Luokkahuoneeseen voi aiheutua meteliä, joka voi haitata parien keskittymistä tehtävään.

Pariohjelmoinnissa tulisi ottaa huomioon parien yhteensopivuus. Tutkimusten mukaan yhteensopivimmat parit ovat opiskelumenestykseltään samalla tasolla (suurin piirtein sama keskiarvo). Parit kannattaa myös jaotella sukupuolen mukaan. Vähemmistöryhmien edustajat kannattaa laittaa pariksi.

Pariohjelmoinnilla on paljon enemmän hyviä puolia kuin huonoja puolia. Pariohjelmointia tulisi lisätä ohjelmoinnin opetuksessa, jotta oppimistulokset paranisivat ja myös naiset saataisiin kiinnostumaan ohjelmoinnista.

Viiteluettelo

Lucy Bain. 2015. Try pair programming. http://developer.atlassian.com/blog/2015/05/try-pair-programming/. Checked 23.10.2015.

Kent Beck. 2000. *Extreme Programming Explained: Embrace Change.* Addison-Wesley.

Tanja Bipp, Andreas Lepper and Doris Schmedding. 2008. Pair programming in software development teams - An empirical study of its benefits. *Information and Software Technology* 50, 3, 231-240.

Gerardo Canfora, Aniello Cimitile and Corrado Aaron Visaggio. 2005. Empirical study on the productivity of the pair programming. In: *Proc. of the 6th International Conference on Extreme Programming and Agile Processes in Software Engineering*. Springer, 92-99.

Timo Gnambs. 2015. What makes a computer wiz? Linking personality traits and programming aptitude. *Journal or Research in Personality* 58, 31-34.

Cecilia M. Gorriz and Claudia Medina. 2000. Engagings girls with computers trough software games. *Communications of the ACM* 43, 1, 42-49.

Brian Hanks. 2005. Student performance in CS1 with distributed pair programming. In: *Proc. of the 10th Annual SIGSCE Conference on Innovation Technology in Computer Science Education*, 316-320.

Brian Hanks. 2006. Student attitudes toward pair programming. In: *ITICSE '06 Proc. of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, 113-117.

Jo E. Hannay, Tore Dybå, Erik Arisholm and Dag I.K. Sjøberg. 2009. The effectiveness of pair programming: A meta-analysis. *Information and Software Technology* 51, 7, 1110-1122.

Neha Katira, Laurie Williams and Jason Osborne. 2005. Towards increasing the compatibility of student pair programmers. In: *ICSE '05 Proc. of the 27th International Conference on Software Engineering*, 625-626.

Kim Man Lui and Keith C.C. Chan. 2006. Pair programming productivity: Novice-novice vs. expert-expert. *International Journal of Human-Computer Studies* 64, 9, 915-925.

Jane Margolis, Allan Fisher and Faye Miller. 1999. Caring about connections: Gender and computing. *IEEE Technology and Society Magazine* 18, 4, 13-20.

Charlie McDowell, Linda Werner, Heather E. Bullock and Julian Fernald. 2006. Pair programming improves student retention, confidence, and program quality. *Communications of the ACM* 49, 8, 90-95.

Daniel Nettle. 2007. *Personality: What Makes You the Way You Are.* Oxford University Press.

David Preston. 2006. Using collaborative learning research to enhance pair programming pedagogy. *ACM SIGITE Newsletter* 3, 1, 16-21.

Norsaremah Salleh, Emilia Mendes, John Grundy and Giles St. J. Burch. 2010. The effects of neuroticism on pair programming: An empirical study in the higher education context. In: *ESEM '10 Proc. of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, Article 22, 10 pages.

Julia Schenk, Lutz Prechelt and Stephan Salinger. 2014. In: *ICSE Companion 2014 Companion Proc. of the 36th International Conference on Software Engineering*, 74-83.

Beth Simon and Brian Hanks. 2008. First-year students' impressions of pair programming in CS1. *Journal on Educational Recourses in Computing* 7, 4, Article 5, 28 pages.

Kelli M. Slaten, Maria Droujkova, Sarah B. Berenson, Laurie Williams and Lucas Layman. 2005. Undergraduate student perceptions of pair programming and agile software methodologies: Verifying a model of social interaction. In: *Agile Conference, 2005. Proceedings. IEEE* 323-330.

Matt Stephens and Doug Rosenberg. 2003. *Extreme Programming Refactored: The Case Against XP.* Apress.

Irfan Naufal Umar and Tie Hui Hui. 2012. Learning style, metaphor and pair programming: Do they influence performance? In: *Proc. of the 4th World Conference on Educational Sciences* 46, 5603-5609.

Tammy VanDeGrift. 2004. Coupling pair programming and writing: Learning about students' perceptions and processes. In: *SIGCSE '13 Proc. of the 44th ACM Technical Symposium on Computer Science Education*, 2-6.

Linda L. Werner, Brian Hanks and Charlie McDowell. 2004. Pair-programming helps female computer science students. *Journal of Educational Resources in Computing 4*, 1, Article 4, 8 pages.

Laurie Williams. 2010. Pair programming. In: *Encyclopedia of Software Engineering*.

L. Williams and R. Kessler. 2003. *Pair Programming Illuminated*. Addison-Wesley.

Laurie Williams and Richard L. Upchurch. 2001. In support of student pair-programming. In: *Proc. of the Thirty-Second SIGCSE Technical Symposium on Computer Science Education*, 327-331.